Advances in Machine Learning: Nearest Neighbour Search, Learning to Optimize and Generative Modelling

by

Ke Li

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Jitendra Malik, Chair
Professor Peter Bartlett
Professor Trevor Darrell
Professor Anant Sahai
Professor Bin Yu

Summer 2019

Advances in Machine Learning: Nearest Neighbour Search, Learning to Optimize and Generative Modelling

Abstract

Advances in Machine Learning: Nearest Neighbour Search, Learning to Optimize and Generative Modelling

by

Ke Li

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Jitendra Malik, Chair

Machine learning is the embodiment of an unapologetically data-driven philosophy that has increasingly become one of the most important drivers of progress in artificial intelligence and beyond. Existing machine learning methods, however, entail making trade-offs in terms of computational efficiency, modelling flexibility and/or formulation faithfulness. In this dissertation, we will cover three different ways in which limitations along each axis can be overcome, without compromising on other axes.

Computational Efficiency

We start with limitations on computational efficiency. Many modern machine learning methods require performing large-scale similarity search under the hood. For example, classifying an input into one of a large number of classes requires comparing the weight vector associated with each class to the activations of the penultimate layer, attending to particular memory cells of a neural net requires comparing the keys associated with each memory cell to the query, and sparse recovery requires comparing each dictionary element to the residual. Similarity search in many cases can be reduced to nearest neighbour search, which is both a blessing and a curse. On the plus side, the nearest neighbour search problem has been extensively studied for more than four decades. On the minus side, no exact algorithm developed over the past four decades can run faster than naïve exhaustive search when the intrinsic dimensionality is high, which is almost certainly the case in machine learning. Given this state of affairs, should we give up any hope of doing any better than the naïve approach of exhaustive comparing each element one-by-one?

It turns out this pessimism, while tempting, is unwarranted. We introduce a new family of *exact* randomized algorithms, known as Dynamic Continuous Indexing, which overcomes both the curse of ambient dimensionality and the curse of intrinsic dimensionality: more

specifically, DCI simultaneously achieves a query time complexity with a *linear* dependence on ambient dimensionality, a *sublinear* dependence on intrinsic dimensionality and a *sublinear* dependence on dataset size. The key insight is that the curse of intrinsic dimensionality in many cases arises from space partitioning, which is a divide-and-conquer strategy used by most nearest neighbour search algorithms. While space partitioning makes intuitive sense and works well in low dimensions, we argue that it fundamentally fails in high dimensions, because it requires distances between each point and every possible query to be approximately preserved in the data structure. We develop a new indexing scheme that only requires the ordering of nearby points relative to distant points to be approximately preserved, and show that the number of out-of-place points after projecting to just a single dimension is sublinear in the intrinsic dimensionality. In practice, our algorithm achieves a $14 - 116\times$ speedup and a $21\times$ reduction in memory consumption compared to locality-sensitive hashing (LSH).

Modelling Flexibility

Next we move onto probabilistic modelling, which is critical to realizing one of the central objectives of machine learning, which is to model the uncertainty that is inherent in prediction. The community has wrestled with the problem of how to strike the right balance between modelling flexibility and computational efficiency. Simple models can often be learned straightforwardly and efficiently but are not expressive; complex models are expressive, but in general cannot be learned both exactly and efficiently, often because learning requires evaluating some intractable integral. The success of deep learning has motivated the development of probabilistic models that can leverage the inductive bias and modelling power of deep neural nets, such as variational autoencoders (VAEs) and generative adversarial nets (GANs), which belong to a subclass of probabilistic models known as implicit probabilistic models. Implicit probabilistic models are defined by a procedure from drawing samples from them, rather than an explicit of the probability density function. On the positive side, sampling is always easy by definition; on the negative side, learning is difficult because not even the unnormalized complete likelihood can be expressed analytically. So these models must be learned using likelihood-free methods, but none have been shown to be able to learn the underlying distribution with a finite number of samples.

Perhaps the most popular likelihood-free method is the GAN. Unfortunately, GANs suffer from the well-documented issue of mode collapse, where the learned model (generator in the GAN parlance) cannot generate some modes of the true data distribution. We argue this arises from the direction in which generated samples are matched to the real data. Under the GAN objective, each *generated sample* is made indistinguishable from some *data example*. Some data examples may not be chosen by any generated sample, resulting in mode collapse. We introduce a new likelihood-free method, known as Implicit Maximum Likelihood Estimation (IMLE) that overcomes mode collapse by inverting the direction - instead of ensuring *each* generated sample has *a* similar data example, our method ensures that *each* data example has *a* similar generated sample. This can be shown to be equivalent to

maximizing a lower bound on the log-likelihood when the model class is richly parameterized and the density is smooth in parameters and data, hence the name.

Compared to VAEs, which are not likelihood-free, IMLE eliminates the need for an approximate posterior and avoids the bias towards parameters where the true posteriors are less informative, a phenomenon known as "posterior collapse".

Formulation Faithfulness

Finally we introduce a novel formulation that can enable the automatic discovery of new iterative gradient-based optimization algorithms, which have become the workhorse of modern machine learning. This effectively allows us to apply machine learning to improve machine learning, which has been a dream of machine learning researchers since the early days of the field. The key challenge, however, is that it is unclear how to represent a complex object like an algorithm in a way that is amenable to machine learning. Prior approaches [58] represent algorithms as imperative programs, i.e.: sequences of elementary operations, and therefore induces a search space whose size is exponential in the length of the optimal program. Searching in this space is unfortunately not tractable for anything but the simplest and shortest algorithms. Other approaches [31] enumerate a small set of manually designed algorithms and search for the best algorithm within this set. Searching in this space is tractable, but the optimal algorithm may lie outside this space. It remains an open question as to how to parameterize the space of possible algorithms in a way that is both *complete* and *efficiently searchable*.

We get around this issue by observing that an optimization algorithm can be uniquely characterized by its update formula – different iterative optimization algorithms only differ in their choice of the update formula. In gradient descent, for example, it is taken to be a scaled negative gradient, whereas in gradient descent with momentum, it is taken to be a scaled exponentially-weighted average of the history of gradients. Therefore, if we can learn the update formula, we can then automatically discover new optimization algorithms. The update formula can be formulated as a mapping from the history of gradients, iterates and objective values to the update step, which can be approximated with a neural net. We can then learn the optimization algorithm by learning the parameters of the neural net.

To my parents, and all other trailblazers of the world.

Contents

List of Figures

List of Tables

Acknowledgments

I cannot believe how quickly time flies and that my PhD journey has come to an end. I know I will look back on this period of my life with fondness and nostalgia, and I would like to take this opportunity to thank all those who have kindly helped me and supported me on this journey.

I am very fortunate to have an AI pioneer and visionary in Prof. Jitendra Malik as my advisor, and am very grateful for all his advice, support, help and wisdom. I am deeply inspired by Jitendra's commitment to objective, thoughtful and rigorous scientific inquiry. I cannot imagine a better advisor than Jitendra.

Throughout my PhD, Jitendra gave me enormous freedom to pursue my interests independently, even when they did not necessarily align with his existing research directions. He trusted me to pick problems to work on and often reassured me that it is fine to try something interesting and fail. This allowed me to take big risks and let my creativity flow, and ultimately led to the three research contributions detailed in this dissertation. Jitendra recognized the significance and potential of each early on, and encouraged me to devote the bulk of my time and energy to them. These projects could not have happened without Jitendra's encouragement and support, for which I would like to thank him immensely.

I would also like to thank my other committee members, Profs. Peter Bartlett, Trevor Darrell, Anant Sahai and Bin Yu, for their helpful feedback and support. I enjoyed the insightful discussions with Prof. Anant Sahai and Prof. Peter Bartlett and was able to gain new perspectives on the fundamentals of machine learning. Moreover, I found their research philosophy and commitment to fundamental research inspiring, and am grateful for their kind help and sage advice both at the tactical and strategic levels.

In addition, I would like to thank all those who have helped me reach where I am now. I remember an inspiring conversation my parents had with me over dinner as I was about to start my PhD. "Einstein made his most important contributions when he was just 26 years old," my father, a theoretical physicist, remarked, "it is never too early to start thinking about how to upend conventional wisdom". My mother, a biochemist, concurred, "it is important to work on something of consequence that you will be proud of decades into the future." I only had four years before I would turn 26, and so from that moment onward, I knew I had to no time to spare and must seize this window of opportunity. I would like to thank my parents, for both challenging and supporting me. Their passion for and devotion to science were a great source of inspiration.

My interest in artificial intelligence (AI) was sparked almost by chance. In the summer of 2008, I was participating in a high school outreach program, known as Computing Insights, run by my local university, the University of Toronto. I was especially intrigued by the title of one lecture, which was on "neural networks" – I had just learned about network flow at the time and wondered what it had to do with the brain. The instructor drew a simple directed graph on the blackboard and explained that it is an abstraction of neurons and synapses. He then switched on his laptop and my attention shifted to the projector screen. Before my eyes, a collection of random pixels gradually transformed into natural-looking handwritten digits.

I was mesmerized. "This is what happens when a neural network dreams," the instructor explained excitedly, "and the digits you see are just figments of the imagination of a deep belief net". I was hooked – I never thought a computer could generate something that looked as if it were produced by a human. "AI doesn't just exist in the realm of science fiction," I thought, "it is *real*." The instructor was none other than Prof. Geoff Hinton, and I would like to thank him for this eye-opening lecture, which ultimately inspired me to study AI.

My foray into research started in the summer of 2013, when Prof. Rich Zemel and Kevin Swersky, kindly agreed to take me under their wings and mentor me on a research project, as part of the NSERC USRA program. Through this experience, I learned about probabilistic models and Perturb-and-MAP at a fundamental level. More importantly, I saw first-hand how existing ideas and techniques can be taken apart and put together in novel ways. I learned a lot from discussions with Rich and Kevin and would like to thank them for their help, guidance and patience. To this day, I recall Kevin's advice that proved valuable: "try to keep an open mind," Kevin told me as I was about to leave Toronto for Berkeley, "and maintain as broad a research interest as possible". I took this advice to heart.

In addition, I am grateful for the intellectually stimulating environment at Berkeley and Toronto that have helped me come up with new ideas. Berkeley and Toronto have a lot in common: a broad array of courses, vigorous debates and long commutes. I benefited tremendously from all of these. The breadth and depth of the course offerings provided me with the background and tools to perform research, the debates at reading groups highlighted the shortcomings of the latest papers, and the long daily commutes and waits at bus stops gave me time and space to ponder deeply. Various elements of these have influenced my research: the ideas for Learning to Optimize and Implicit Maximum Likelihood Estimation (IMLE) were born during a bus ride and a flight home respectively, the idea for Dynamic Continuous Indexing (DCI) was born out of a project for a course on multi-armed bandits taught by Prof. Peter Bartlett, the analytical techniques used for DCI were inspired by a course on randomized algorithms taught by Prof. Lap Chi Lau, the switch in the underlying technique of Learning to Optimize to reinforcement learning was inspired by two courses, one on deep reinforcement learning taught by John Schulman and one on robotics taught by Prof. Pieter Abbeel, the motivation for working on DCI originated with a (perhaps facetious) comment by Prof. Alyosha Efros that all of machine learning can be reduced to nearest neighbours, and the motivation for working on IMLE originated with Prof. Christos Papadimitriou's computational hardness results of finding Nash equilibria.

In the later years of my PhD, I discovered the flip side of overturning conventional wisdom – it is hard, sometimes dauntingly so. Coming up with a radically new idea and demonstrating its practical utility is not the end of the journey, but rather the start. Moreover, the more unconventional the idea, the more tortuous the journey becomes. Completing this journey successfully takes more than just the soundness of the idea or the abundance of evidence; it takes courage, determination and perseverance. This journey is the process of turning a manuscript into a peer-reviewed publication. There were times when I felt hopeless and was on the verge of giving up. But I never felt alone. I looked to the scientists I admire. I recalled the article Jitendra shared on how eight groundbreaking papers in physics, chemistry and

biology were initially rejected but later won the Nobel Prize. I recalled Prof. Geoff Hinton's story on how his paper on semantic hashing was rejected because two other deep learning papers have already been accepted for publication. I recalled Prof. Shafi Goldwasser's story on how her paper on zero-knowledge proofs was rejected because of concerns on whether the use of the term "proof" was warranted. I recalled my mother's story on how her paper on DNA cleavage through hydrolysis rather than oxidation was rejected because of a widely held belief that it was not possible. I would like to thank these scientists for sharing their very inspiring stories and for giving me the strength and confidence to press on in the face of hardship.

Finally, I would like to thank my friends, labmates, colleagues, instructors, mentees, TAs and readers. I thoroughly enjoyed my time at Berkeley and the privilege of working with and alongside so many talented individuals. Special shoutouts go to Weicheng Kuo, Deepak Pathak, Christian Häne, Shubham Tulsiani, Saurabh Gupta, Pulkit Agrawal, David Fouhey, Tianhao Zhang, Shichong Peng and Kailas Vodrahalli. Last but not least, I would like to thank you, my dear reader, for reading this dissertation. Perhaps you are just getting started in your scientific career or are new to the field, and I hope you will derive value from reading it. While you are reading this, I would like to present you with a challenge: find something wrong with the scientific consensus of your time, and work to fix it. Nothing in science should be accepted at face value simply because it is popular and nothing should be dismissed because it goes against the mainstream; in fact, the greatest breakthroughs come from questioning things that are broadly accepted by the scientific community. As scientists, we owe our present understanding to the trailblazers of the past, and the only way to give back is to strive to be trailblazers ourselves. While this is not the easiest path to take, it is the most fulfilling. To all who choose this path, I salute you – you can count me in as a supporter and I hope you will succeed.

Chapter 1

Nearest Neighbour Search

The method of k-nearest neighbours is a fundamental building block of many machine learning algorithms and also has broad applications beyond artificial intelligence, including in statistics, bioinformatics and database systems, e.g. [28, 15, 50]. The problem statement is simple: Given a set of n points, $S = \{p^1, \ldots, p^n\} \subseteq \mathbb{R}^d$ and a query point $q \in \mathbb{R}^d$, the goal is to find k vectors in S that are the closest to q in Euclidean distance.

Since the method of nearest neighbour search was introduced by Fix & Hodges [55] in 1951, it has for decades intrigued the artificial intelligence and theoretical computer science communities alike. In low dimensions (think < 10 dimensions), devising efficient sublinear algorithms, i.e.: algorithms that permit querying in time sublinear in the dataset size, seems easy – many early algorithms, like k-d trees [21], achieve a query time complexity that is logarithmic in the dataset size. This propelled the success of many methods for computational geometry.

In machine learning, however, the dimensionality of data is typically much higher, i.e.: on the order of hundreds or more. Unfortunately, repeated attempts at devising exact sublinear algorithms over the past four decades have encountered a recurring obstacle: *the curse of dimensionality*. That is, the query time complexity or space complexity always has an *exponential* dependence on dimensionality. As a result, practitioners often have resort to naïve exhaustive search, which entails iterating over all the points in the dataset, computing the distance from the query to each and then returning the top-k elements.

Is it possible to overcome the curse of dimensionality? Minsky and Papert [99] conjectured that doing so is impossible, and the best we can hope for in high dimensions is exhaustive search. Later work [80] showed that if we were to differentiate between ambient dimensionality, which characterizes properties of the ambient space, and intrinsic dimensionality, which characterizes properties of the data, then it is possible to overcome the curse of *ambient* dimensionality. But it remains an open question whether the curse of *intrinsic* dimensionality can be overcome – all exact algorithms that can overcome the curse of ambient dimensionality suffer from the curse of intrinsic dimensionality.

In this dissertation, we answer this question in the affirmative and show that it is in fact possible to overcome both the curse of ambient dimensionality and the curse of intrin-

sic dimensionality. We introduce a new family of exact randomized algorithms, known as Dynamic Continuous Indexing (DCI), that can achieve this – its query time complexity is simultaneously sublinear in the dataset size, linear in the ambient dimensionality and sublinear in the intrinsic dimensionality. Additionally, we show empirically that the proposed algorithm reduces the number of distance evaluations by a factor of 14 to 116 and the memory consumption by a factor of 21 relative to Locality-Sensitive Hashing (LSH).

1.1 Notions of Dimensionality

Two notions of dimensionality are commonly considered. The more familiar notion, ambient dimensionality, refers to the dimensionality of the space data points are embedded in. In other words, it is simply the ordinary notion of dimensionality, and is so named so that we can differentiate it with the next notion of dimensionality we are about to cover.

On the other hand, intrinsic dimensionality characterizes the intrinsic properties of the data and measures the rate at which the number of points inside a ball grows as a function of its radius. Note that there are several different measures of intrinsic dimensionality – the one we use here is arguably the simplest and most commonly used in the literature, namely the expansion dimension or KR-dimension introduced by Karger & Ruhl [80]. More precisely:

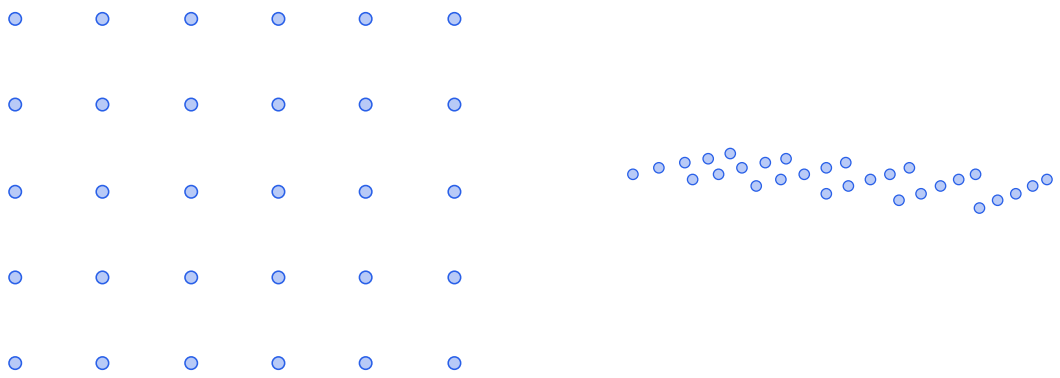
Definition 1 (Expansion Dimension [80])**.** *Given a dataset $D \subseteq \mathbb{R}^d$, let $B_p(r)$ be the set of points in D that are within a ball of radius r around a point p. A dataset D has (τ, d')-expansion if for all r and $p \in \mathbb{R}^d$ such that $|B_p(r)| \geq \tau$, $|B_p(2r)| \leq 2^{d'} |B_p(r)|$. The quantity of interest we are interested is the smallest possible value of d' when $\tau = k$, where k is the number of nearest neighbours we would like to retrieve. This quantity is known as the* expansion dimension*, or simply the* intrinsic dimensionality *in our context.*

So, for a dataset with intrinsic dimensionality d', any ball of radius r contains at most $O(r^{d'})$ points. If the data points are arranged on a uniform grid in a d'-dimensional subspace, then the intrinsic dimensionality is exactly d'. So, roughly speaking, for data points that are uniformly distributed, then the intrinsic dimensionality is usually close to the dimensionality of the manifold, as long as the manifold is sufficiently smooth. Two datasets with same intrinsic dimensionality and different dimensionalities are shown in Figure 1.1.

1.2 Exact vs. Approximate Nearest Neighbour Search

In the exact version of the k-nearest neighbour search problem, the goal is to return the k closest data points to the query.

Due to difficulties of devising efficient algorithms for exact version of the problem in high dimensions, various methods consider the approximate version, which relaxes the requirements for the returned results to be declared correct. Under the approximate setting, it is permissible to return data points whose distances to the query are within a factor of $(1 + \epsilon)$ of the distance between the query and the kth nearest neighbour.



(a) (b)

Figure 1.1: An illustration of two datasets with the same intrinsic dimensionality, 2, but different ambient dimensionalities. Figure (a) shows a dataset with an ambient dimensionality of 2, and Figure (b) shows a dataset with an ambient dimensionality of 3.

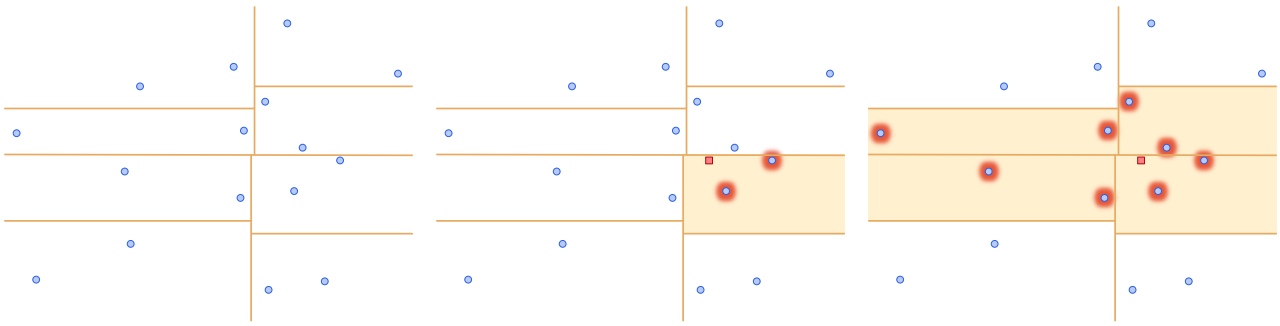
1.3 Space Partitioning

Space partitioning is a popular divide-and-conquer strategy that forms the basis of most existing methods for nearest neighbour search. It works by partitioning the vector space into a finite number of cells and keeps track of the data points that each cell contains. At query time, these methods simply look up of the contents of the cell containing the query and possibly adjacent cells and perform exhaustive search over points lying in these cells. Methods that are based on space partitioning include k-d trees [21] and locality-sensitive hashing (LSH) [76]. We discuss below how k-d trees and LSH work concretely and why they can be viewed as space partitioning-based methods.

1.3.1 k-d Trees

The k-d tree was proposed by Jon Bentley in 1975 and is a deterministic tree-based data structure that is commonly used for exact nearest neighbour search. It works by recursively dividing the dataset into halves along different axes, until the number of points in each cell falls below a threshold. More concretely, at each node in the tree, we select an axis and pick a threshold along that axis such that half of the data points associated with the node falls on the left, and the other half falls on the right. The left child of the node is associated with the data points to the left of the threshold, and the right child is associated with the remaining data points. A visualization of what the resulting partitioning looks like in 2D is shown in Figure 1.2a.

At query time, we traverse the tree to find the cell that contains the query and search over the data points in that cell, as shown in Figure 1.2b. If the neighbouring cells could



(a) (b) (c)

Figure 1.2: An illustration of how a k-d tree partitions the space in the two-dimensional case. Figure (a) shows the partitioning performed by a k-d tree, where blue circles denote data points and yellow lines denote cell boundaries. Each line is located at the threshold used at a particular internal node to divide the dataset and each cell corresponds to a leaf node, Figure (b) shows the data points (highlighted in red) in the cell containing the given query (shown as the red square) and Figure (c) shows the data points in neighbouring cells, which all need to be searched.

contain data points that are closer to those found so far, we go back up the tree to find the neighbouring cells, and then search over data points lying in those cells as well, as shown in Figure 1.2c.

Why is it necessary to look in neighbouring cells? If the query lies near a cell boundary, there could be data points on the other side of the cell boundary that are closer to the query than any of the data points in the cell containing the query.

Now consider what happens in the worst case. There is a configuration of data points and query such that the query is very close to a cell boundary in all dimensions, forcing the algorithm to search all neighbouring cells. When the ambient dimensionality increases, the number of neighbouring cells could grow exponentially. This is how the curse of ambient dimensionality arises – the number of neighbouring cells grows exponentially in the ambient dimensionality, thereby leading to a query time complexity that is exponential in ambient dimensionality.

It seems like this worst case is somewhat contrived, and so we can try to avoid it using randomization. This is the idea behind RP trees [40]. Unlike k-d trees, the directions of the dividing hyperplanes are not necessarily aligned to axes; instead, they are randomly chosen from a unit sphere. This improves the dependence of query time complexity from exponential in the *ambient* dimensionality to exponential in the *intrinsic* dimensionality, which could be much lower than ambient dimensionality. Intuitively, if we pick a random partitioning, there are usually many cells that are away from the manifold and do not contain data points. Hence it is safe to avoid searching these cells. While this eliminates the curse of ambient dimensionality, it does not get around the curse of intrinsic dimensionality.

1.3.2 Locality-Sensitive Hashing

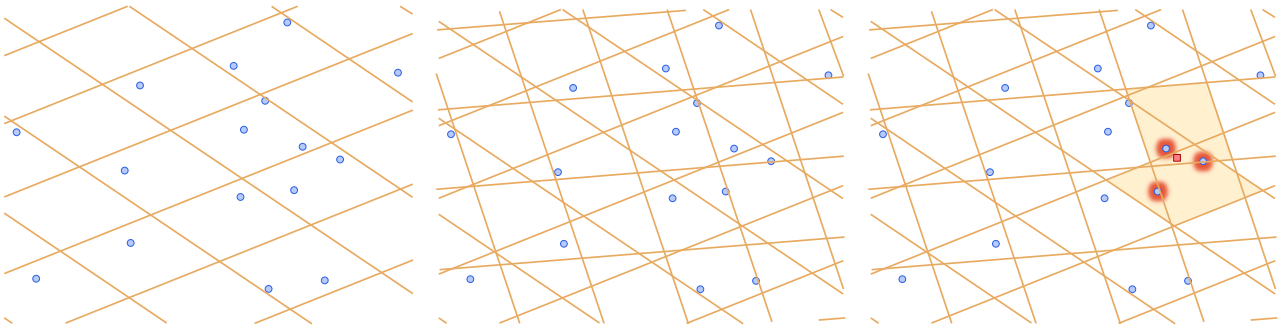


Figure 1.3: An illustration of how Euclidean LSH partitions the space in the two-dimensional case. Blue circles denote data points, yellow lines denote cell boundaries. Figure (a) shows the partitioning imposed by one hash table, Figure (b) shows the partitioning imposed by two hash tables, and Figure (c) shows the data points to search over exhaustively (which are highlighted in red) for the given query (shown as the red square).

Locality-sensitive hashing (LSH) was proposed by Piotr Indyk and Rajeev Motwani in 1998 that is commonly used for approximate nearest neighbour search. Euclidean LSH [42] uses a hash function that can be applied to vectors in Euclidean space, and can be used to perform approximate nearest neighbour search in Euclidean space. At construction time, we project all data points along a direction randomly chosen from the unit sphere and then place them into equal-sized discrete bins based on the values of data points after projection, which will be referred to as projection values. We repeat this multiple times, each with a different random projection direction. The hash associated with a data point is the vector of bin indices, each of which corresponds to a projection direction. Then, for each distinct hash value, we store all the data points that hash to this value in a hash table. We construct multiple different hash tables, each generated using different collections of random projection directions.

At query time, for each hash table, we compute the hash of the query and look up the data points associated with the hash. We then search over the union of the retrieved data points, over all hash tables and return the k data points that are closest to the query.

Geometrically, projecting data points along a random direction and then discretizing the projection values into equal-sized bins is equivalent to dividing the vector space into equal-sized randomly-oriented parallel slabs and keeping track of the data points that lie in each slab. Treating the concatenation of bin indices as the hash effectively further subdivides the slabs according to the values of bin indices along other projection directions, and so all the points that have the same hash must be in the intersection of the slabs corresponding to different projection directions. Therefore, each hash table effectively performs space partitioning using a regular grid, as shown in Figure 1.3a. Different hash tables correspond

to different overlapping grids, as shown in Figure 1.3b. The data points that we search over are all the data points that lie in any of the cells that contain the query, as shown in Figure 1.3c.

1.4 Landscape of Prior Methods

Many other algorithms for nearest neighbour search have been proposed, and below we survey the landscape of nearest neighbour search methods. It is by no means exhaustive; interested readers may refer to [36] for a more comprehensive survey.

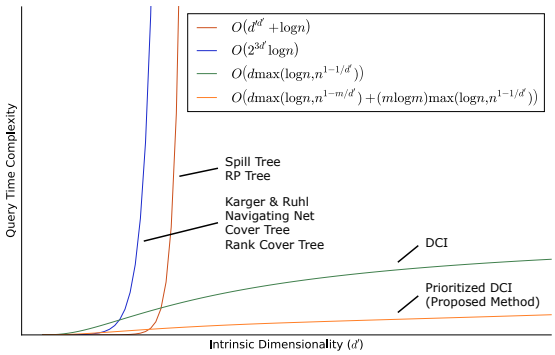


Figure 1.4: Visualization of the query time complexities of various exact algorithms as a function of the intrinsic dimensionality d'. Each curve represents an example from a class of similar query time complexities. Algorithms that fall into each particular class are shown next to the corresponding curve.

Nearest neighbour search algorithms can be divided into two categories: algorithms that solve the exact version of the problem, known as exact algorithms, and algorithms that solve the approximate version, known as approximate algorithms. See Section 1.2 for the distinction between exact and approximate versions of the problem. Approximate algorithms are not to be confused with randomized algorithms: randomized algorithms must return the correct result with high probability, and whereas approximate algorithms broadens the criterion for correctness by permitting solutions that are *nearly* as good, but not necessarily as good, as the exact solution. Both exact randomized algorithms and approximate randomized algorithms are possible – the former refers to an algorithm that returns the exact solution with high probability, whereas the latter refers to an algorithm that returns a solution that is suboptimal by at most a factor of $(1+\epsilon)$ with high probability.

Early exact algorithms are deterministic and store points in tree-based data structures. Examples include k-d trees, R-trees [64] and X-trees [22, 23], which divide the vector space into a hierarchy of half-spaces, hyper-rectangles or Voronoi polygons and keep track of the points that lie in each cell. While their query times are logarithmic in the size of the dataset, they exhibit exponential dependence on the ambient dimensionality. A different method [98] partitions the space by intersecting multiple hyperplanes. It effectively trades off space for time and achieves polynomial query time in ambient dimensionality at the cost of exponential space complexity in ambient dimensionality.

To avoid poor performance on worst-case configurations of the data, exact randomized algorithms have been proposed. Spill trees [95], RP trees [40] and virtual spill trees [41] extend

Method	Query Time Complexity
Exact Algorithms:	
RP Tree	$O((d' \log d')^{d'} + \log n)$
Spill Tree	$O(d'^{d'} + \log n)$
[80]	$O(2^{3d'} \log n)$
Navigating Net	$2^{O(d')} \log n$
Cover Tree	$O(2^{12d'} \log n)$
Rank Cover Tree	$O(2^{O(d' \log h)} n^{2/h})$ for $h \geq 3$
DCI (Ours)	$O(d(m + \max(\log n, n^{1-1/d'})))$
Prioritized DCI (Ours)	$O(d(m + \max(\log n, n^{1-m/d'}) + (m \log m) \max(\log n, n^{1-1/d'})))$ for some $m \geq 1$ chosen by the user
Approximate Algorithms:	
k-d Tree	$O((1/\epsilon)^d \log n)$
BBD Tree	$O((6/\epsilon)^d \log n)$
LSH	$\approx O(dn^{1/(1+\epsilon)^2})$

Table 1.1: Query time complexities of various algorithms for 1-NN search. Ambient dimensionality, intrinsic dimensionality, dataset size and approximation ratio are denoted as d, d', n and $1 + \epsilon$. A visualization of the growth of various time complexities as a function of the intrinsic dimensionality is shown in Figure 1.4.

the ideas behind k-d trees by randomizing the orientations of hyperplanes that partition the space into half-spaces at each node of the tree. While randomization enables them to avoid exponential dependence on the ambient dimensionality, their query times still scale exponentially in the intrinsic dimensionality. Whereas these methods rely on space partitioning, other algorithms [105, 35, 80] have been proposed that utilize local search strategies. These methods start with a random point and look in the neighbourhood of the current point to find a new point that is closer to the query than the original in each iteration. Like space partitioning-based approaches, the query time of [80] scales exponentially in the intrinsic dimensionality. While the query times of [105, 35] do not exhibit such undesirable dependence, their space complexities are quadratic in the size of the dataset, making them impractical for large datasets. A different class of algorithms performs search in a coarse-to-fine manner. Examples include navigating nets [82], cover trees [27] and rank cover trees [73], which maintain sets of subsampled data points at different levels of granularity and descend through the hierarchy of neighbourhoods of decreasing radii around the query. Unfortunately, the query times of these methods again scale exponentially in the intrinsic dimensionality.

Many of the same strategies are employed by approximate algorithms. Methods based on tree-based space partitioning [11] and local search [10] have been developed; like many exact algorithms, their query times also scale exponentially in the ambient dimensionality. Locality-Sensitive Hashing (LSH) [76, 42, 2] partitions the space into regular cells, whose shapes are implicitly defined by the choice of the hash function. It achieves a query time of $O(dn^{\rho})$ using $O(dn^{1+\rho})$ space, where d is the ambient dimensionality, n is the dataset

size and $\rho \approx 1/(1+\epsilon)^2$ for large n in Euclidean space, though the dependence on intrinsic dimensionality is not made explicit. In practice, the performance of LSH degrades on datasets with large variations in density, due to the uneven distribution of points across cells. Consequently, various data-dependent hashing schemes have been proposed [106, 125, 3]; unlike data-independent hashing schemes, however, they do not allow dynamic updates to the dataset. A related approach [78] decomposes the space into mutually orthogonal axis-aligned subspaces and independently partitions each subspace. It has a query time linear in the dataset size and no known guarantee on the probability of correctness under the exact or approximate setting. A different approach [1] projects the data to a lower dimensional space that approximately preserves approximate nearest neighbour relationships and applies other approximate algorithms like BBD trees [11] to the projected data. Its query time is also linear in ambient dimensionality and sublinear in the dataset size. Unlike LSH, it uses space linear in the dataset size, at the cost of longer query time than LSH. Unfortunately, its query time is exponential in intrinsic dimensionality.

A summary of the query times of various prior algorithms and the proposed algorithm is presented in Table 1.1 and their growth as a function of intrinsic dimensionality is illustrated in Figure 1.4.

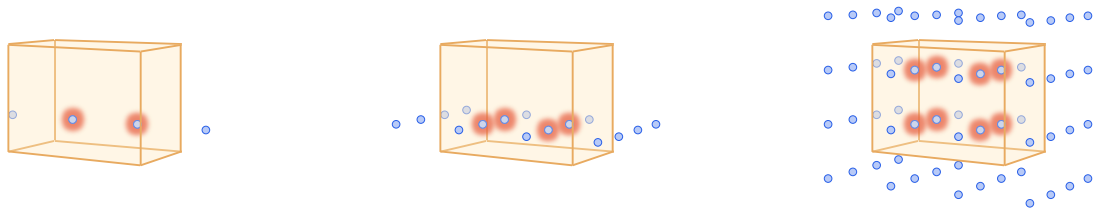
1.5 Curse of Intrinsic Dimensionality

All exact sublinear algorithms suffer from the curse of ambient dimensionality or the curse of intrinsic dimensionality, that is, their query time complexity or space complexity have exponential dependence on ambient or intrinsic dimensionality. As explained in Section 1.3.1, while the curse of ambient dimensionality can be overcome, it is still unclear how to overcome the curse of intrinsic dimensionality. We discuss below why the curse of intrinsic dimensionality arises.

Consider any data-independent partitioning scheme and focus on a cell in the partitioning. We ask the following question: how many data points could there be inside the cell as the intrinsic dimensionality increases?

We consider canonical examples of datasets with integer dimensionalities, namely data points arranged on a uniform grid in a subspace. Take an example of a partitioning where each cell is a cube. Then as shown in Figure 1.5, as the intrinsic dimensionality increases, the number of points inside the cell increases exponentially. This shows that the number of points inside a cell of the shape of a cube could grow exponentially in the worst case (over the choice of dataset). Because we can fit a cube inside a non-empty cell of any shape and rescale the dataset arbitrarily without changing the intrinsic dimensionality, the number of points inside a cell of *any* shape could grow exponentially as the intrinsic dimensionality increase.

This implies that regardless of the data-independent partitioning we choose, the number of data points inside a cell could grow exponentially in intrinsic dimensionality. Because we need to exhaustively search over all data points within a cell in a space partitioning-



(a) $d' = 1$ (b) $d' = 2$ (c) $d' = 3$

Figure 1.5: The number of data points that lie inside a cell in a data-independent partitioning, which is shown as the yellow cube, as the intrinsic dimensionality increases. Figures (a), (b) and (c) show three canonical examples of datasets with intrinsic dimensionalities d' of 1, 2 and 3 respectively. As shown, as the intrinsic dimensionality increases, the number of data points that lie inside the cell grows exponentially.

based method, we will always have an exponential dependence on intrinsic dimensionality in the query time complexity. So, to have any hope of overcoming the curse of intrinsic dimensionality, we must avoid data-independent space partitioning.

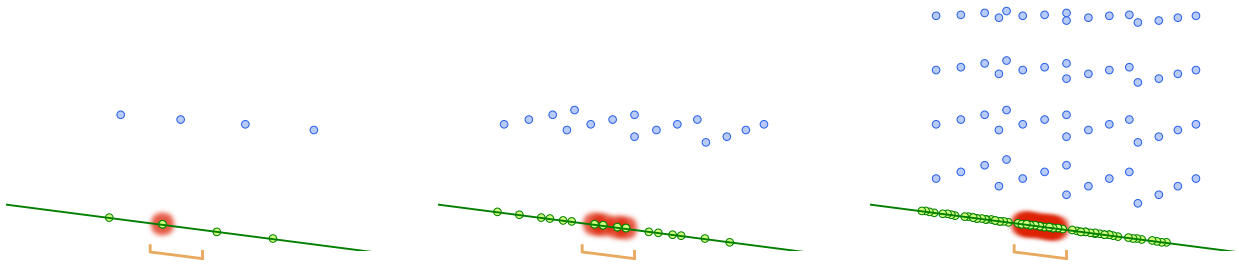
1.6 Key Insight

We will first discuss the key insight behind the proposed family of algorithms, known as Dynamic Continuous Indexing (DCI), before describing the details in later sections.

As shown in Section 1.5, data-independent space partitioning causes the curse of intrinsic dimensionality, and so we would like to eliminate it.

The first step we take is to eliminate discretization after projecting data points along a random direction. Without discretization, it is no longer clear which data points should be retrieved given a query, since there is no bin from which we can look up data points. Instead, let's consider a natural alternative: we can project the query along the projection direction, and then retrieve the data points that project to a neighbourhood of a certain radius around the query projection. As shown in Figure 1.6, for *any* neighbourhood of a fixed size, as the intrinsic dimensionality increases, the number of data points whose projections are in the neighbourhood could grow exponentially. So, eliminating space partitioning is itself not enough to overcome the curse of intrinsic dimensionality. We must also avoid choosing a fixed-size neighbourhood.

To this end, instead of retrieving all data points within a fixed *distance*, we retrieve a fixed *number* of data points. (This can be equivalently viewed as choosing a neighbourhood whose radius is dependent on the query and the data rather than fixed.) More concretely, starting from the query projection, we will march along the projection direction and retrieve

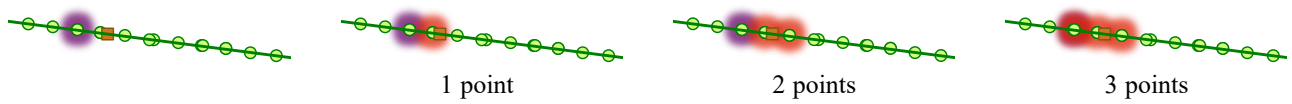


(a) $d' = 1$ (b) $d' = 2$ (c) $d' = 3$

Figure 1.6: Naïve approach of avoiding space partitioning, which projects all data points onto a random direction and looks at a neighbourhood of a certain radius around the query along the projection direction. Figures (a), (b) and (c) three canonical examples of datasets with intrinsic dimensionalities d' of 1, 2 and 3 respectively. The green line denotes the projection direction, the yellow bracket denotes the neighbourhood around the query along the projection direction, the green points on the line denote projections of data points onto the direction and the green points that are highlighted in red denote points in the neighbourhood. As shown, as the intrinsic dimensionality increases, the number of points within the neighbourhood grows exponentially.

data points in the order of increasing distance to the query along the projection direction. Figure 1.7 illustrates each step of this process; in the example shown, the true nearest neighbour is contained within the first three data points. In general, it turns out that the nearest neighbour must be contained within the first $O(n^{1-1/d'})$ points with constant probability, where n denotes the number of data points and d' denotes the intrinsic dimensionality. (The proof of this result is shown in the appendix.) The advantage of this approach comes from the dependence of this function on the intrinsic dimensionality – it is sublinear and no longer exponential.

There is a significant difference in the goal of the data structure compared to methods based on space partitioning. In methods based on space partitioning, the goal is to approximately preserve the absolute locations of data points, so that data points that are nearby would be close in the data structure. On the other hand, the goal of the proposed indexing scheme is to approximately preserve the relative *order* between the true nearest neighbours and the other data points when ranked by their distances to the query. In other words, we care about distortions in ranks, rather than distortions in absolute locations. In other words, we would be happy if a few distant points seem close to the true nearest neighbours, because while distortions in absolute locations are high, distortions in ranks are low, because the true nearest neighbours' positions in the ranking only moved down by a few places. On the other hand, if our goal were to preserve the absolute locations, we would not be happy. Similarly, we can also contrast our goal with that of dimensionality reduction methods like the Johnson-Lindenstrauss transform. In Johnson-Lindenstrauss, the goal is to approxi-



(a) (b) (c) (d)

Figure 1.7: Retrieval of data points in the order of increasing distance from the query along the projection direction. Figures (a), (b), (c) and (d) show the data points that are retrieved after the zeroth through the fourth iteration. The brown square denotes the projection of the query, the the green line denotes the projection direction, the green points on the line denote projections of data points onto the direction, the green point that is highlighted in purple denotes the projection of the true nearest neighbour to the query, and the green points that are highlighted in red denote points that are retrieved. In this case, we are able to retrieve the correct nearest neighbour within three points. In general, the nearest neighbour must be encountered within the first $O(n^{1-1/d'})$ points with constant probability.

mately preserve all pairwise distances, which is why we need to project to at least $\Omega(\log(n))$ dimensions. Because our goal is weaker, we can get away with projecting to one dimension.

1.7 Generalized Union Bound

The following result is used in the analysis of the proposed algorithms, which may be of independent interest. It is a generalization of the union bound, and upper bounds the probability that k out of a set of possibly dependent events happen. When $k = 1$, it reduces to the union bound. The proof is in the appendix.

Lemma 1. *For any set of events* $\{E_i\}_{i=1}^N$*, the probability that at least* k' *of them happen is at most* $\frac{1}{k'} \sum_{i=1}^N \Pr(E_i)$.

1.8 Dynamic Continuous Indexing (DCI)

DCI constructs a data structure consisting of multiple *composite indices* of data points, each of which in turn consists of a number of *simple indices*. Each simple index orders data points according to their projections along a particular random direction. Given a query, for every composite index, the algorithm finds points that are near the query in every constituent simple index, which are known as *candidate points*, and adds them to a set known as the *candidate set*. The true distances from the query to every candidate point are evaluated and the ones that are among the k closest to the query are returned.

More concretely, each simple index is associated with a random direction and stores the projections of every data point along the direction. They are implemented using standard data structures that maintain one-dimensional ordered sequences of elements, like self-balancing binary search trees [14, 61] or skip lists [107]. At query time, the algorithm projects the query along the projection directions associated with each simple index and finds the position where the query would have been inserted in each simple index, which takes logarithmic time. It then iterates over, or *visits*, data points in each simple index in the order of their distances to the query under projection, which takes constant time for each iteration. As it iterates, it keeps track of how many times each data point has been visited across all simple indices of each composite index. If a data point has been visited in every constituent simple index, it is added to the candidate set and is said to have been *retrieved* from the composite index. A precise statement of the construction and querying procedures are shown in Algorithms 3 and 4.

Algorithm 1 Data structure construction procedure

Require: A dataset D of n points $p^1, \ldots, p^n$, the number of simple indices m that constitute a composite index and the number of composite indices L

function CONSTRUCT(D, m, L)
  $\{u_{jl}\}_{j\in[m],l\in[L]} \leftarrow mL$ random unit vectors in $\mathbb{R}^d$
  $\{T_{jl}\}_{j\in[m],l\in[L]} \leftarrow mL$ empty binary search trees or skip lists
  for $j = 1$ **to** m **do**
    for $l = 1$ **to** L **do**
      for $i = 1$ **to** n **do**
        $\overline{p}^i_{jl} \leftarrow \langle p^i, u_{jl}\rangle$
        Insert $(\overline{p}^i_{jl}, i)$ into T_{jl} with $\overline{p}^i_{jl}$ being the key and i being the value
      end for
    end for
  end for
  return $\{(T_{jl}, u_{jl})\}_{j\in[m],l\in[L]}$
end function

DCI has a number of appealing properties compared to methods based on space partitioning. Because points are visited by rank rather than absolute location in space, DCI performs well on datasets with large variations in data density. It naturally skips over sparse regions of the space and concentrates more on dense regions of the space. Since construction of the data structure does not depend on the dataset, the algorithm supports dynamic updates to the dataset, while being able to automatically adapt to changes in data density. Furthermore, because data points are represented in the indices as continuous values without being discretized, the granularity of discretization does not need to be chosen at construction time. Consequently, the same data structure can support queries at varying desired levels of accuracy, which allows a different speed-vs-accuracy trade-off to be made for each individual query.

We develop two versions of the algorithm, a data-independent and a data-dependent version, which differ in the stopping condition that is used. In the former, the number of

Algorithm 2 k-nearest neighbour retrieval procedure

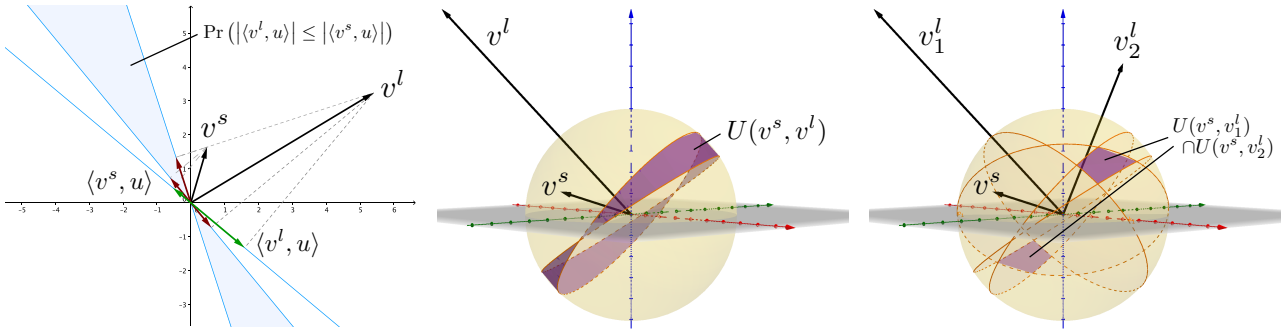
Require: Query point q in $\mathbb{R}^d$, binary search trees/skip lists and their associated projection vectors $\{(T_{jl}, u_{jl})\}_{j\in[m],l\in[L]}$, and maximum tolerable failure probability ϵ
function QUERY($q, \{(T_{jl}, u_{jl})\}_{j,l}, \epsilon$)
  $C_l \leftarrow$ array of size n with entries initialized to 0 $\forall l \in [L]$
  $\overline{q}_{jl} \leftarrow \langle q, u_{jl}\rangle \; \forall j \in [m], l \in [L]$
  $S_l \leftarrow \emptyset \; \forall l \in [L]$
  for $i = 1$ **to** n **do**
    for $l = 1$ **to** L **do**
      for $j = 1$ **to** m **do**
        $(\overline{p}_{jl}^{(i)}, h_{jl}^{(i)}) \leftarrow$ the node in T_{jl} whose key is the i^{th} closest to $\overline{q}_{jl}$
        $C_l[h_{jl}^{(i)}] \leftarrow C_l[h_{jl}^{(i)}] + 1$
      end for
      for $j = 1$ **to** m **do**
        if $C_l[h_{jl}^{(i)}] = m$ **then**
          $S_l \leftarrow S_l \cup \{h_{jl}^{(i)}\}$
        end if
      end for
    end for
    if *IsStoppingConditionSatisfied*(i, S_l, ϵ) **then**
      break
    end if
  end for
  return k points in $\bigcup_{l\in[L]} S_l$ that are the closest in Euclidean distance in $\mathbb{R}^d$ to q
end function

candidate points is indirectly preset according to the global data density and the maximum tolerable failure probability; in the latter, the number of candidate points is chosen adaptively at query time based on the local data density in the neighbourhood of the query. We analyze the algorithm below and show that its query time complexity is linear in ambient dimensionality, sublinear in intrinsic dimensionality and sublinear in the size of the dataset. In addition, we show that its space complexity is independent of ambient dimensionality and linear in the size of the dataset.

1.8.1 Analysis

We now analyze the time and space complexities of the algorithm. For proofs of the following results, see the appendix.

First, we examine the effect of projecting d-dimensional vectors to one dimension, which motivates its use in the proposed algorithm. We are interested in the probability that a distant point appears closer than a nearby point under projection; if this probability is low, then each simple index approximately preserves the order of points by distance to the query point. If we consider displacement vectors between the query point and data points, this probability is then is equivalent to the probability of the lengths of these vectors inverting



(a) (b) (c)

Figure 1.8: (a) Examples of order-preserving (shown in green) and order-inverting (shown in red) projection directions. Any projection direction within the shaded region inverts the relative order of the vectors by length under projection, while any projection directions outside the region preserves it. The size of the shaded region depends on the ratio of the lengths of the vectors. (b) Projection vectors whose endpoints lie in the shaded region would be order-inverting. (c) Projection vectors whose endpoints lie in the shaded region would invert the order of both long vectors relative to the short vector. Best viewed in colour.

Property	Complexity
Construction	$O(m(dn + n\log n))$
Query	$O(\max(d(m + k\log(n/k), dk(n/k)^{1-1/d'})))$
Insertion	$O(m(d + \log n))$
Deletion	$O(m\log n)$
Space	$O(mn)$

Table 1.2: Time and space complexities of DCI.

under projection.

Lemma 2. *Let $v^l, v^s \in \mathbb{R}^d$ such that $\|v^l\|_2 > \|v^s\|_2$, and $u \in \mathbb{R}^d$ be a unit vector drawn uniformly at random. Then the probability of v^s being at least as long as v^l under projection u is at most $1 - \frac{2}{\pi}\cos^{-1}\left(\|v^s\|_2 / \|v^l\|_2\right)$.*

Observe that if $\left|\langle v^l, u\rangle\right| \leq \left|\langle v^s, u\rangle\right|$, the relative order of v^l and v^s by their lengths would be inverted when projected along u. This occurs when $u^{\|}$ is close to orthogonal to v^l, which is illustrated in Figure 1.8a. Also note that the probability of inverting the relative order of v^l and v^s is small when v^l is much longer than v^s. On the other hand, this probability is high when v^l and v^s are similar in length, which corresponds to the case when two data points are almost equidistant to the query point. So, if we consider a sequence of vectors ordered by length, applying random one-dimensional projection will likely perturb the ordering locally, but will preserve the ordering globally.

Next, we build on this result to analyze the order-inversion probability when there are more than two vectors. Consider the sample space $B = \left\{ u \in \mathbb{R}^d \middle| \|u\|_2 = 1 \right\}$ and the set $U(v^s, v^l) = \left\{ u \in B \middle| |\cos\theta| \leq \left\|v^s\right\|_2 / \left\|v^l\right\|_2 \right\}$, which is illustrated in Figure 1.8b, where θ is the angle between $u^{\|}$ and v^l. If we use area(U) to denote the area of the region formed by the endpoints of all vectors in the set U, then we can rewrite the above bound on the order-inversion probability as:

$$\begin{aligned}\Pr\left(\left|\langle v^l, u\rangle\right| \leq \left|\langle v^s, u\rangle\right|\right) &\leq \Pr\left(u \in U(v^s, v^l)\right) \\ &= \frac{\text{area}(V_l^s)}{\text{area}(B)} \\ &= 1 - \frac{2}{\pi}\cos^{-1}\left(\frac{\|v^s\|_2}{\|v^l\|_2}\right)\end{aligned}$$

Lemma 3. *Let $\left\{v_i^l\right\}_{i=1}^N$ be a set of vectors such that $\left\|v_i^l\right\|_2 > \left\|v^s\right\|_2 \ \forall i \in [N]$. Then the probability that there is a subset of k' vectors from $\left\{v_i^l\right\}_{i=1}^N$ that are all not longer than v^s under projection is at most $\frac{1}{k'}\sum_{i=1}^N \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v^s\right\|_2 / \left\|v_i^l\right\|_2\right)\right)$. Furthermore, if $k' = N$, this probability is at most $\min_{i\in[N]}\left\{1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v^s\right\|_2 / \left\|v_i^l\right\|_2\right)\right\}$.*

Intuitively, if this event occurs, then there are at least k' vectors that rank above v^s when sorted in nondecreasing order by their lengths under projection. This can only occur when the endpoint of u falls in a region on the unit sphere corresponding to $\bigcup_{I\subseteq[N]:|I|=k'}\bigcap_{i\in I} U(v^s, v_i^l)$. We illustrate this region in Figure 1.8c for the case of $d = 3$.

Theorem 1. *Let $\left\{v_i^l\right\}_{i=1}^N$ and $\left\{v_{i'}^s\right\}_{i'=1}^{N'}$ be sets of vectors such that $\left\|v_i^l\right\|_2 > \|v_{i'}^s\|_2 \forall i \in [N], i' \in [N']$. Then the probability that there is a subset of k' vectors from $\left\{v_i^l\right\}_{i=1}^N$ that are all not longer than some $v_{i'}^s$ under projection is at most $\frac{1}{k'}\sum_{i=1}^N \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v_{\max}^s\right\|_2 / \left\|v_i^l\right\|_2\right)\right)$, where $\left\|v_{\max}^s\right\|_2 \geq \|v_{i'}^s\|_2 \forall i' \in [N']$.*

We define the following derivative quantities of the intrinsic dimensionality, which are easier to work with in our context:

Definition 2. *Given a dataset $D \subseteq \mathbb{R}^d$, let $B_p(r)$ be the set of points in D that are within a ball of radius r around a point p. We say D has local relative sparsity of (τ, γ) at a point $p \in \mathbb{R}^d$ if for all r such that $|B_p(r)| \geq \tau$, $|B_p(\gamma r)| \leq 2\,|B_p(r)|$, where $\gamma \geq 1$.*

Intuitively, γ represents a lower bound on the increase in radius when the number of points within the ball is doubled. When γ is close to 1, the dataset is dense in the neighbourhood of p, since there could be many points in D that are almost equidistant from p. Retrieving the nearest neighbours of such a p is considered "hard", since it would be difficult to tell which of these points are the true nearest neighbours without computing the distances to all these points exactly.

We also define a related notion of global relative sparsity, which we will use to derive the number of iterations the outer loop of the querying function should be executed and a bound on the running time that is independent of the query:

Definition 3. *A dataset D has global relative sparsity of (τ, γ) if for all r and $p \in \mathbb{R}^d$ such that $|B_p(r)| \geq \tau$, $|B_p(\gamma r)| \leq 2\,|B_p(r)|$, where $\gamma \geq 1$.*

Note that a dataset with global relative sparsity of (τ, γ) has local relative sparsity of (τ, γ) at every point. Global relative sparsity is closely related to the notion of *expansion rate* introduced by [80]. More specifically, a dataset with global relative sparsity of (τ, γ) has $(\tau, 1/\log_2 \gamma)$-expansion, where the latter quantity is the expansion dimension, also known as the intrinsic dimensionality. So, the intrinsic dimensionality of a dataset with global relative sparsity of (τ, γ) is $1/\log_2 \gamma$.

1.8.2 Data-Independent Version

In the data-independent version of the algorithm, the outer loop in the querying function executes for a preset number of iterations $\tilde{k}$. The values of L, m and $\tilde{k}$ are fixed for all queries and will be chosen later.

We apply the results obtained above to analyze the algorithm. Consider the event that the algorithm fails to return the correct set of k-nearest neighbours – this can only occur if a true k-nearest neighbour is not contained in any of the S_l's, which entails that for each $l \in [L]$, there is a set of $\tilde{k} - k + 1$ points that are not the true k-nearest neighbours but are closer to the query than the true k-nearest neighbours under some of the projections $u_{1l}, \ldots, u_{ml}$. We analyze the probability that this occurs below and derive the parameter settings that ensure the algorithm succeeds with high probability.

Lemma 4. *For a dataset with global relative sparsity (k, γ), there is some $\tilde{k} \in \Omega(\max(k \log(n/k), k(n/k)^{1-\log_2 \gamma}))$ such that the probability that the candidate points retrieved from a given composite index do not include some of the true k-nearest neighbours is at most some constant $\alpha < 1$.*

Theorem 2. *For a dataset with global relative sparsity (k, γ), for any $\epsilon > 0$, there is some L and $\tilde{k} \in \Omega(\max(k \log(n/k), k(n/k)^{1-\log_2 \gamma}))$ such that the algorithm returns the correct set of k-nearest neighbours with probability of at least $1 - \epsilon$.*

The above result suggests that we should choose $\tilde{k} \in \Omega(\max(k \log(n/k), k(n/k)^{1-\log_2 \gamma}))$ to ensure the algorithm succeeds with high probability. Next, we analyze the time and space complexity of the algorithm.

Theorem 3. *The algorithm takes $O(\max(d(m + k \log(n/k), dk(n/k)^{1-1/d'})))$ time to retrieve the k-nearest neighbours at query time, where d' denotes the intrinsic dimensionality of the dataset.*

Theorem 4. *The algorithm takes $O(dn + n \log n)$ time to preprocess the data points in D at construction time.*

Theorem 5. *The algorithm requires $O(d + \log n)$ time to insert a new data point and $O(\log n)$ time to delete a data point.*

Theorem 6. *The algorithm requires $O(n)$ space in addition to the space used to store the data.*

1.8.3 Data-Dependent Version

Conceptually, performance of the proposed algorithm depends on two factors: how likely the index returns the true nearest neighbours before other points and when the algorithm stops retrieving points from the index. The preceding sections primarily focused on the former; in this section, we take a closer look at the latter.

One strategy, which is used by the data-independent version of the algorithm, is to stop after a preset number of iterations of the outer loop. Although simple, such a strategy leaves much to be desired. First of all, in order to set the number of iterations, it requires knowledge of the global relative sparsity of the dataset, which is rarely known a priori. Computing this is either very expensive in the case of datasets or infeasible in the case of streaming data, as global relative sparsity may change as new data points arrive. More importantly, it is unable to take advantage of the local relative sparsity in the neighbourhood of the query. A method that is capable of adapting to local relative sparsity could potentially be much faster because query points tend to be close to the manifold on which points in the dataset lie, resulting in the dataset being sparse in the neighbourhood of the query point.

Ideally, the algorithm should stop as soon as it has retrieved the true nearest neighbours. Determining if this is the case amounts to asking if there exists a point that we have not seen lying closer to the query than the points we have seen. At first sight, because nothing is known about unseen points, it seems not possible to do better than exhaustive search, as we can only rule out the existence of such a point after computing distances to all unseen points. Somewhat surprisingly, by exploiting the fact that the projections associated with the index are random, it is possible to make inferences about points that we have never seen. We do so by leveraging ideas from statistical hypothesis testing.

After each iteration of the outer loop, we perform a hypothesis test, with the null hypothesis being that the complete set of the k-nearest neighbours has not yet been retrieved. Rejecting the null hypothesis implies accepting the alternative hypothesis that all the true k-nearest neighbours have been retrieved. At this point, the algorithm can safely terminate while guaranteeing that the probability that the algorithm fails to return the correct results is bounded above by the significance level. The test statistic is an upper bound on the probability of missing a true k-nearest neighbour. The resulting algorithm does not require any prior knowledge about the dataset and terminates earlier when the dataset is sparse in the neighbourhood of the query; for this reason, we will refer to this version of the algorithm as the data-dependent version.

More concretely, as the algorithm retrieves candidate points, it computes their true distances to the query and maintains a list of k points that are the closest to the query among the points retrieved from all composite indices so far. Let $\tilde{p}^{(i)}$ and $\tilde{p}_l^{\max}$ denote the i^{th} closest candidate point to q retrieved from all composite indices and the farthest candidate point from q retrieved from the l^{th} composite index respectively. When the number of candidate points

exceeds k, the algorithm checks if $\prod_{l=1}^{L}\left(1-\left(\frac{2}{\pi}\cos^{-1}\left(\left\|\tilde{p}^{(k)}-q\right\|_2 / \left\|\tilde{p}_l^{\max}-q\right\|_2\right)\right)^m\right) \leq \epsilon$, where ϵ is the maximum tolerable failure probability, after each iteration of the outer loop. If the condition is satisfied, the algorithm terminates and returns $\left\{\tilde{p}^{(i)}\right\}_{i=1}^{k}$.

We show the correctness and running time of this algorithm below.

Theorem 7. *For any $\epsilon > 0$, m and L, the data-dependent algorithm returns the correct set of k-nearest neighbours of the query q with probability of at least $1-\epsilon$.*

Theorem 8. *On a dataset with global relative sparsity (k, γ), given fixed parameters m and L, the data-dependent algorithm takes $O\left(\max\left(dk\log\left(\frac{n}{k}\right), dk\left(\frac{n}{k}\right)^{1-1/d'}, \frac{d}{\left(1-\sqrt[m]{1-\sqrt[L]{\epsilon}}\right)^{d'}}\right)\right)$ time with high probability to retrieve the k-nearest neighbours at query time, where d' denotes the intrinsic dimensionality of the dataset.*

Note that we can make the denominator of the last argument arbitrarily close to 1 by choosing a large L.

1.9 Prioritized DCI

Prioritized DCI differs from standard DCI in the order in which points from different simple indices are visited. In standard DCI, the algorithm cycles through all constituent simple indices of a composite index at regular intervals and visits exactly one point from each simple index in each pass. In Prioritized DCI, the algorithm assigns a priority to each constituent simple index; in each iteration, it visits the upcoming point from the simple index with the highest priority and updates the priority at the end of the iteration. The priority of a simple index is set to the negative absolute difference between the query projection and the next data point projection in the index.

Intuitively, this ensures data points are visited in the order of their distances to the query under projection. Because data points are only retrieved from a composite index when they have been visited in all constituent simple indices, data points are retrieved in the order of the maximum of their distances to the query along multiple projection directions. Since distance under projection forms a lower bound on the true distance, the maximum projected distance approaches the true distance as the number of projection directions increases. Hence, in the limit as the number of simple indices approaches infinity, data points are retrieved in the ideal order, that is, the order of their true distances to the query.

The construction and querying procedures of Prioritized DCI are presented formally in Algorithms 3 and 4. To ensure the algorithm retrieves the exact k-nearest neighbours with high probability, the analysis in the next section shows that one should choose $k_0 \in \Omega(k\max(\log(n/k), (n/k)^{1-m/d'}))$ and $k_1 \in \Omega(mk\max(\log(n/k), (n/k)^{1-1/d'}))$, where d' denotes the intrinsic dimensionality. Though because this assumes worst-case configuration of data points, it may be overly conservative in practice; so, these parameters may be chosen by cross-validation.

Algorithm 3 Data structure construction procedure

Require: A dataset D of n points $p^1, \ldots, p^n$, the number of simple indices m that constitute a composite index and the number of composite indices L

function CONSTRUCT(D, m, L)
  $\{u_{jl}\}_{j\in[m],l\in[L]} \leftarrow mL$ random unit vectors in $\mathbb{R}^d$
  $\{T_{jl}\}_{j\in[m],l\in[L]} \leftarrow mL$ empty binary search trees or skip lists
  for $j = 1$ **to** m **do**
    for $l = 1$ **to** L **do**
      for $i = 1$ **to** n **do**
        $\overline{p}^i_{jl} \leftarrow \langle p^i, u_{jl}\rangle$
        Insert $(\overline{p}^i_{jl}, i)$ into T_{jl} with $\overline{p}^i_{jl}$ being the key and i being the value
      end for
    end for
  end for
  return $\{(T_{jl}, u_{jl})\}_{j\in[m],l\in[L]}$
end function

Property	Complexity
Construction	$O(m(dn + n\log n))$
Query	$O\left(d(m + k\max(\log(n/k), (n/k)^{1-m/d'}) + mk\log m(\max(\log(n/k), (n/k)^{1-1/d'}))\right)$
Insertion	$O(m(d + \log n))$
Deletion	$O(m\log n)$
Space	$O(mn)$

Table 1.3: Time and space complexities of Prioritized DCI.

We summarize the time and space complexities of Prioritized DCI in Table 1.3. Notably, the first term of the query complexity, which dominates when the ambient dimensionality d is large, has a more favourable dependence on the intrinsic dimensionality d' than the query complexity of standard DCI. In particular, a linear increase in the intrinsic dimensionality, which corresponds to an exponential increase in the expansion rate, can be mitigated by just a linear increase in the number of simple indices m. This suggests that Prioritized DCI can better handle datasets with high intrinsic dimensionality than standard DCI, which is confirmed by empirical evidence later in this paper.

1.9.1 Analysis

We analyze the time and space complexities of Prioritized DCI below and derive the stopping condition of the algorithm. Because the algorithm uses standard data structures, analysis of the construction time, insertion time, deletion time and space complexity is straightforward. Hence, this section focuses mostly on analyzing the query time. For proofs, see the appendix.

In high-dimensional space, query time is dominated by the time spent on evaluating

Algorithm 4 k-nearest neighbour querying procedure

Require: Query point q in $\mathbb{R}^d$, binary search trees/skip lists and their associated projection vectors $\{(T_{jl}, u_{jl})\}_{j\in[m],l\in[L]}$, the number of points to retrieve k_0 and the number of points to visit k_1 in each composite index

function QUERY($q, \{(T_{jl}, u_{jl})\}_{j,l}, k_0, k_1$)

$C_l \leftarrow$ array of size n with entries initialized to 0 $\forall l \in [L]$

$\overline{q}_{jl} \leftarrow \langle q, u_{jl}\rangle \; \forall j \in [m], l \in [L]$

$S_l \leftarrow \emptyset \; \forall l \in [L]$

$P_l \leftarrow$ empty priority queue $\forall l \in [L]$

for $l = 1$ **to** L **do**

for $j = 1$ **to** m **do**

$(\overline{p}_{jl}^{(1)}, h_{jl}^{(1)}) \leftarrow$ the node in T_{jl} whose key is the closest to $\overline{q}_{jl}$

Insert $(\overline{p}_{jl}^{(1)}, h_{jl}^{(1)})$ with priority $-|\overline{p}_{jl}^{(1)} - \overline{q}_{jl}|$ into P_l

end for

end for

for $i' = 1$ **to** $k_1 - 1$ **do**

for $l = 1$ **to** L **do**

if $|S_l| < k_0$ **then**

$(\overline{p}_{jl}^{(i)}, h_{jl}^{(i)}) \leftarrow$ the node with the highest priority in P_l

Remove $(\overline{p}_{jl}^{(i)}, h_{jl}^{(i)})$ from P_l and insert the node in T_{jl} whose key is the next closest to $\overline{q}_{jl}$, which is denoted as $(\overline{p}_{jl}^{(i+1)}, h_{jl}^{(i+1)})$, with priority $-|\overline{p}_{jl}^{(i+1)} - \overline{q}_{jl}|$ into P_l

$C_l[h_{jl}^{(i)}] \leftarrow C_l[h_{jl}^{(i)}] + 1$

if $C_l[h_{jl}^{(i)}] = m$ **then**

$S_l \leftarrow S_l \cup \{h_{jl}^{(i)}\}$

end if

end if

end for

end for

return k points in $\bigcup_{l\in[L]} S_l$ that are the closest in Euclidean distance in $\mathbb{R}^d$ to q

end function

true distances between candidate points and the query. Therefore, we need to find the number of candidate points that must be retrieved to ensure the algorithm succeeds with high probability. To this end, we derive an upper bound on the failure probability for any given number of candidate points. The algorithm fails if sufficiently many distant points are retrieved from each composite index before some of the true k-nearest neighbours. We decompose this event into multiple (dependent) events, each of which is the event that a particular distant point is retrieved before some true k-nearest neighbours. Since points are retrieved in the order of their maximum projected distance, this event happens when the maximum projected distance of the distant point is less than that of a true k-nearest neighbour. We start by finding an upper bound on the probability of this event. To simplify notation, we initially consider displacement vectors from the query to each data point, and so relationships between projected distances of triplets of points translate relationships between projected lengths of pairs of displacement vectors.

We start by examining the event that a vector under random one-dimensional projection satisfies some geometric constraint. We then find an upper bound on the probability that some combinations of these events occur, which is related to the failure probability of the algorithm.

Lemma 5. *Let $v^l, v^s \in \mathbb{R}^d$ be such that $\left\|v^l\right\|_2 > \left\|v^s\right\|_2$, $\left\{u'_j\right\}_{j=1}^M$ be i.i.d. unit vectors in $\mathbb{R}^d$ drawn uniformly at random. Then* $\Pr\left(\max_j \left\{\left|\langle v^l, u'_j\rangle\right|\right\} \leq \left\|v^s\right\|_2\right) = \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v^s\right\|_2 / \left\|v^l\right\|_2\right)\right)^M$.

Combining the above yields the following theorem.

Theorem 9. *Let $\left\{v^l_i\right\}_{i=1}^N$ and $\left\{v^s_{i'}\right\}_{i'=1}^{N'}$ be sets of vectors such that $\left\|v^l_i\right\|_2 > \left\|v^s_{i'}\right\|_2$ $\forall i \in [N], i' \in [N']$. Furthermore, let $\left\{u'_{ij}\right\}_{i\in[N],j\in[M]}$ be random uniformly distributed unit vectors such that $u'_{i1}, \ldots, u'_{iM}$ are independent for any given i. Consider the events $\left\{\exists v^s_{i'} \text{ s.t. } \max_j \left\{\left|\langle v^l_i, u'_{ij}\rangle\right|\right\} \leq \left\|v^s_{i'}\right\|_2\right\}_{i=1}^N$. The probability that at least k' of these events occur is at most $\frac{1}{k'}\sum_{i=1}^N \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v^s_{\max}\right\|_2 / \left\|v^l_i\right\|_2\right)\right)^M$, where $\left\|v^s_{\max}\right\|_2 = \max_{i'}\left\{\left\|v^s_{i'}\right\|_2\right\}$. Furthermore, if $k' = N$, it is at most $\min_{i\in[N]}\left\{\left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v^s_{\max}\right\|_2 / \left\|v^l_i\right\|_2\right)\right)^M\right\}$.*

We now apply the results above to analyze specific properties of the algorithm. For convenience, instead of working directly with intrinsic dimensionality, we will analyze the query time in terms of a related quantity, global relative sparsity, as defined in [89]. We reproduce its definition below for completeness.

Definition 4. *Given a dataset $D \subseteq \mathbb{R}^d$, let $B_p(r)$ be the set of points in D that are within a ball of radius r around a point p. A dataset D has global relative sparsity of (τ, γ) if for all r and $p \in \mathbb{R}^d$ such that $|B_p(r)| \geq \tau$, $|B_p(\gamma r)| \leq 2\,|B_p(r)|$, where $\gamma \geq 1$.*

Global relative sparsity is related to the expansion rate [80] and intrinsic dimensionality in the following way: a dataset with global relative sparsity of (τ, γ) has $(\tau, 2^{(1/\log_2\gamma)})$-expansion and intrinsic dimensionality of $1/\log_2\gamma$.

Below we derive two upper bounds on the probability that some of the true k-nearest neighbours are missing from the set of candidate points retrieved from a given composite index, which are in expressed in terms of k_0 and k_1 respectively. These results inform us how k_0 and k_1 should be chosen to ensure the querying procedure returns the correct results with high probability. In the results that follow, we use $\{p^{(i)}\}_{i=1}^n$ to denote a re-ordering of the points $\{p^i\}_{i=1}^n$ so that $p^{(i)}$ is the i^{th} closest point to the query q.

Lemma 6. *Consider points in the order they are retrieved from a composite index that consists of m simple indices. The probability that there are at least n_0 points that are not the true k-nearest neighbours but are retrieved before some of them is at most* $\frac{1}{n_0-k}\sum_{i=2k+1}^{n}\left(1-\frac{2}{\pi}\cos^{-1}\left(\left\|p^{(k)}-q\right\|_2/\left\|p^{(i)}-q\right\|_2\right)\right)^m$.

Lemma 7. *Consider point projections in a composite index that consists of m simple indices in the order they are visited. The probability that there are n_0 point projections that are not the true k-nearest neighbours but are visited before all true k-nearest neighbours have been retrieved is at most* $\frac{m}{n_0-mk}\sum_{i=2k+1}^{n}\left(1-\frac{2}{\pi}\cos^{-1}\left(\left\|p^{(k)}-q\right\|_2/\left\|p^{(i)}-q\right\|_2\right)\right)$.

Lemma 8. *On a dataset with global relative sparsity (k,γ), the quantity* $\sum_{i=2k+1}^{n}\left(1-\frac{2}{\pi}\cos^{-1}\left(\left\|p^{(k)}-q\right\|_2/\left\|p^{(i)}-q\right\|_2\right)\right)^m$ *is at most* $O\left(k\max(\log(n/k),(n/k)^{1-m\log_2\gamma})\right)$.

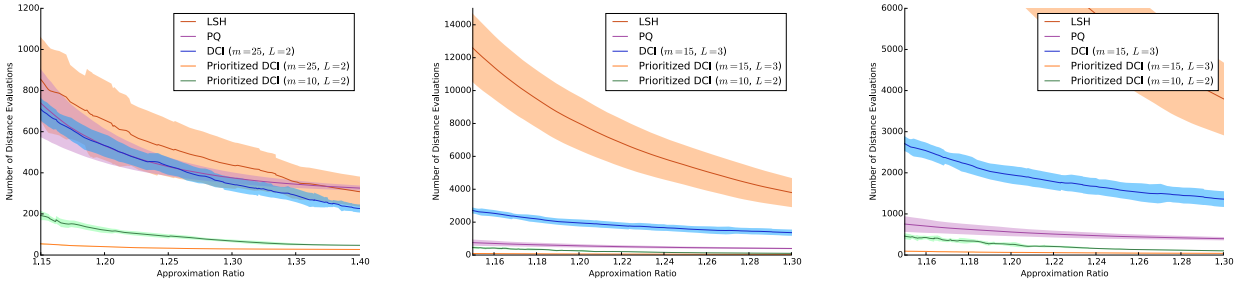
Lemma 9. *For a dataset with global relative sparsity (k,γ) and a given composite index consisting of m simple indices, there is some $k_0\in\Omega(k\max(\log(n/k),(n/k)^{1-m\log_2\gamma}))$ such that the probability that the candidate points retrieved from the composite index do not include some of the true k-nearest neighbours is at most some constant $\alpha_0<1$.*

Lemma 10. *For a dataset with global relative sparsity (k,γ) and a given composite index consisting of m simple indices, there is some $k_1\in\Omega(mk\max(\log(n/k),(n/k)^{1-\log_2\gamma}))$ such that the probability that the candidate points retrieved from the composite index do not include some of the true k-nearest neighbours is at most some constant $\alpha_1<1$.*

Theorem 10. *For a dataset with global relative sparsity (k,γ), for any $\epsilon>0$, there is some L, $k_0\in\Omega(k\max(\log(n/k),(n/k)^{1-m\log_2\gamma}))$ and $k_1\in\Omega(mk\max(\log(n/k),(n/k)^{1-\log_2\gamma}))$ such that the algorithm returns the correct set of k-nearest neighbours with probability of at least $1-\epsilon$.*

Now that we have found a choice of k_0 and k_1 that suffices to ensure correctness with high probability, we can derive a bound on the query time that guarantees correctness. We then analyze the time complexity for construction, insertion and deletion and the space complexity.

Theorem 11. *For a given number of simple indices m, the algorithm takes* $O\left(d(m+k\max(\log(n/k),(n/k)^{1-m/d'}))+mk\log m\left(\max(\log(n/k),(n/k)^{1-1/d'})\right)\right)$ *time to retrieve the k-nearest neighbours at query time, where d' denotes the intrinsic dimensionality.*



(a) (b) (c)

Figure 1.9: Comparison of the number of distance evaluations needed by different algorithms to achieve varying levels of approximation quality on (a) CIFAR-100 and (b,c) MNIST. Each curve represents the mean over ten folds and the shaded area represents ± 1 standard deviation. Lower values are better. (c) Close-up view of the figure in (b).

Theorem 12. *For a given number of simple indices m, the algorithm takes $O(m(dn + n \log n))$ time to preprocess the data points in D at construction time.*

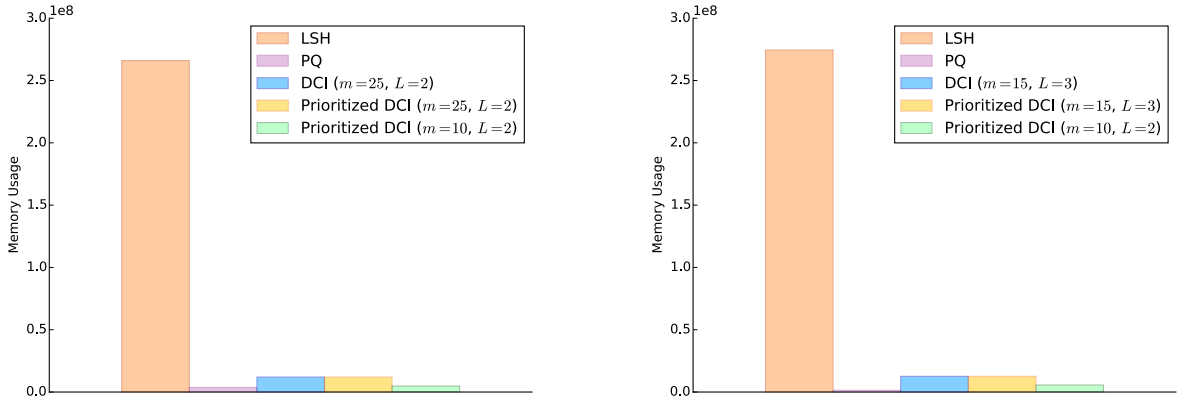
Theorem 13. *The algorithm requires $O(m(d + \log n))$ time to insert a new data point and $O(m \log n)$ time to delete a data point.*

Theorem 14. *The algorithm requires $O(mn)$ space in addition to the space used to store the data.*

1.10 Experiments

We compare the performance of Prioritized DCI to that of standard DCI [89], product quantization [78] and LSH [42], which is perhaps the algorithm that is most widely used in high-dimensional settings. Because LSH operates under the approximate setting, in which the performance metric of interest is how close the returned points are to the query rather than whether they are the true k-nearest neighbours. All algorithms are evaluated in terms of the time they would need to achieve varying levels of approximation quality.

Evaluation is performed on two datasets, CIFAR-100 [83] and MNIST [85]. CIFAR-100 consists of $60,000$ colour images of 100 types of objects in natural scenes and MNIST consists of $70,000$ grayscale images of handwritten digits. The images in CIFAR-100 have a size of 32×32 and three colour channels, and the images in MNIST have a size of 28×28 and a single colour channel. We reshape each image into a vector whose entries represent pixel intensities at different locations and colour channels in the image. So, each vector has a dimensionality of $32 \times 32 \times 3 = 3072$ for CIFAR-100 and $28 \times 28 = 784$ for MNIST. Note that the dimensionalities under consideration are much higher than those typically used to evaluate prior methods.



(a) (b)

Figure 1.10: Memory usage of different algorithms on (a) CIFAR-100 and (b) MNIST. Lower values are better.

For the purposes of nearest neighbour search, MNIST is a more challenging dataset than CIFAR-100. This is because images in MNIST are concentrated around a few modes; consequently, data points form dense clusters, leading to higher intrinsic dimensionality. On the other hand, images in CIFAR-100 are more diverse, and so data points are more dispersed in space. Intuitively, it is much harder to find the closest digit to a query among 6999 other digits of the same category that are all plausible near neighbours than to find the most similar natural image among a few other natural images with similar appearance. Later results show that all algorithms need fewer distance evaluations to achieve the same level of approximation quality on CIFAR-100 than on MNIST.

We evaluate performance of all algorithms using cross-validation, where we randomly choose ten different splits of query vs. data points. Each split consists of 100 points from the dataset that serve as queries, with the remainder designated as data points. We use each algorithm to retrieve the 25 nearest neighbours at varying levels of approximation quality and report mean performance and standard deviation over all splits.

Approximation quality is measured using the approximation ratio, which is defined to be the ratio of the radius of the ball containing the set of true k-nearest neighbours to the radius of the ball containing the set of approximate k-nearest neighbours returned by the algorithm. The closer the approximation ratio is to 1, the higher the approximation quality. In high dimensions, the time taken to compute true distances between the query and the candidate points dominates query time, so the number of distance evaluations can be used as an implementation-independent proxy for the query time.

For LSH, we used 24 hashes per table and 100 tables, which we found to achieve the best approximation quality given the memory constraints. For product quantization, we used a data-independent codebook with 256 entries so that the algorithm supports dynamic

updates. For standard DCI, we used the same hyperparameter settings used in [89] ($m = 25$ and $L = 2$ on CIFAR-100 and $m = 15$ and $L = 3$ on MNIST). For Prioritized DCI, we used two different settings: one that matches the hyperparameter settings of standard DCI, and another that uses less space ($m = 10$ and $L = 2$ on both CIFAR-100 and MNIST).

We plot the number of distance evaluations that each algorithm requires to achieve each desired level of approximation ratio in Figure 1.9. As shown, on CIFAR-100, under the same hyperparameter setting used by standard DCI, Prioritized DCI requires 87.2% to 92.5% fewer distance evaluations than standard DCI, 91.7% to 92.8% fewer distance evaluations than product quantization, and 90.9% to 93.8% fewer distance evaluations than LSH to achieve same levels approximation quality, which represents a 14-fold reduction in the number of distance evaluations relative to LSH on average. Under the more space-efficient hyperparameter setting, Prioritized DCI achieves a 6-fold reduction compared to LSH. On MNIST, under the same hyperparameter setting used by standard DCI, Prioritized DCI requires 96.4% to 97.0% fewer distance evaluations than standard DCI, 87.1% to 89.8% fewer distance evaluations than product quantization, and 98.8% to 99.3% fewer distance evaluations than LSH, which represents a 116-fold reduction relative to LSH on average. Under the more space-efficient hyperparameter setting, Prioritized DCI achieves a 32-fold reduction compared to LSH.

We compare the space efficiency of Prioritized DCI to that of standard DCI and LSH. As shown in Figure 1.10, compared to LSH, Prioritized DCI uses 95.5% less space on CIFAR-100 and 95.3% less space on MNIST under the same hyperparameter settings used by standard DCI. This represents a 22-fold reduction in memory consumption on CIFAR-100 and a 21-fold reduction on MNIST. Under the more space-efficient hyperparameter setting, Prioritized DCI uses 98.2% less space on CIFAR-100 and 97.9% less space on MNIST relative to LSH, which represents a 55-fold reduction on CIFAR-100 and a 48-fold reduction on MNIST.

In terms of wall-clock time, our implementation of Prioritized DCI takes 1.18 seconds to construct the data structure and execute 100 queries on MNIST, compared to 104.71 seconds taken by LSH.

Chapter 2

Learning to Optimize

Machine learning has enjoyed tremendous success and is being applied to a wide variety of areas, both in AI and beyond. This success can be attributed to the data-driven philosophy that underpins machine learning, which favours automatic discovery of patterns from data over manual design of systems using expert knowledge.

Yet, there is a paradox in the current paradigm: the algorithms that power machine learning are still designed manually. This raises a natural question: can we learn these algorithms instead? This could open up exciting possibilities: we could find new algorithms that perform better than manually designed algorithms, which could in turn improve learning capability.

Doing so, however, requires overcoming a fundamental obstacle: how do we parameterize the space of algorithms so that it is both (1) expressive, and (2) efficiently searchable? Various ways of representing algorithms trade off these two goals. For example, if the space of algorithms is represented by a small set of known algorithms, it most likely does not contain the best possible algorithm, but does allow for efficient searching via simple enumeration of algorithms in the set. On the other hand, if the space of algorithms is represented by the set of all possible programs, it contains the best possible algorithm, but does not allow for efficient searching, as enumeration would take exponential time.

One of the workhorses of machine learning is continuous optimization algorithms; more broadly, they are some of the most ubiquitous tools used in virtually all areas of science and engineering. Several popular algorithms exist, including gradient descent, momentum, AdaGrad and ADAM. We consider the problem of automatically designing such algorithms. Why do we want to do this? There are two reasons: first, many optimization algorithms are devised under the assumption of convexity and applied to non-convex objective functions; by learning the optimization algorithm under the same setting as it will actually be used in practice, the learned optimization algorithm could hopefully achieve better performance. Second, devising new optimization algorithms manually is usually laborious and can take months or years; learning the optimization algorithm could reduce the amount of manual labour.

Algorithm	Update Formula π
Gradient Descent	$\pi(\cdot) = -\gamma \nabla f(x^{(i-1)})$
Momentum	$\pi(\cdot) = -\gamma \left(\sum_{j=0}^{i-1} \alpha^{i-1-j} \nabla f(x^{(j)}) \right)$
Conjugate Gradient	$\pi(\cdot) = -\gamma \left(\nabla f(x^{(i-1)}) + \sum_{j=0}^{i-2} \left(\frac{\left\|\nabla f(x^{(j+1)})\right\|^2}{\left\|\nabla f(x^{(j)})\right\|^2} \right)^{i-1-j} \nabla f(x^{(j)}) \right)$

Table 2.1: Choices of the update formula π made by hand-engineered optimization algorithms. We propose learning π automatically in the hope of learning an optimization algorithm that converges faster and to better optima on objective functions of interest.

2.1 Formulation

Consider how existing continuous optimization algorithms generally work. As outlined in Algorithm 5, they operate in an iterative fashion and maintain some iterate $x^{(i)}$, which is a point in the domain of the objective function. Initially, the iterate is some random point in the domain; in each iteration, a step vector Δx is computed using some fixed update formula, which is then used to modify the iterate. The update formula π is some functional of the objective function, the current iterate and past iterates. Typically, it is some function of the history of gradients of the objective function evaluated at the current and past iterates. For example, in gradient descent, the update formula is some scaled negative gradient; in momentum, the update formula is some scaled exponential moving average of the gradients.

Algorithm 5 General structure of unconstrained optimization algorithms

Require: Objective function f
  $x^{(0)} \leftarrow$ random point in the domain of f
  for $i = 1, 2, \ldots$ **do**
    $\Delta x \leftarrow \pi(f, \{x^{(0)}, \ldots, x^{(i-1)}\})$
    if stopping condition is met **then**
      return $x^{(i-1)}$
    end if
    $x^{(i)} \leftarrow x^{(i-1)} + \Delta x$
  end for

What changes from algorithm to algorithm is this update formula π. Examples of existing optimization algorithms and their corresponding update formulas are shown in Table 2.1.

So, if we can learn the update formula π, we will be able to learn an optimization algorithm. Since it is difficult to model general functionals, in practice, we restrict the dependence of π on the objective function f to objective values and gradients evaluated at current and past iterates. Hence, π can be simply modelled as a function from the objective

values and gradients along the trajectory taken by the optimizer so far to the next step vector. If we model π with a universal function approximator like a neural net, it is then possible to search over the space of optimization algorithms by learning the parameters of the neural net.

Parameterizing the update formula as a neural net has two appealing properties mentioned earlier: first, it is expressive, as neural nets are universal function approximators and can in principle model any update formula with sufficient capacity; second, it allows for efficient search, as neural nets can be trained easily with backpropagation.

In order to learn the optimization algorithm, we need to define a performance metric, which we will refer to as the "meta-loss", that rewards good optimizers and penalizes bad optimizers. Since a good optimizer converges quickly, a natural meta-loss would be the sum of objective values over all iterations (assuming the goal is to minimize the objective function), or equivalently, the cumulative regret. Intuitively, this corresponds to the area under the curve, which is larger when the optimizer converges slowly and smaller otherwise.

2.2 Learning How to Learn

When the objective functions under consideration correspond to loss functions for training a model, the proposed framework effectively learns how to learn. The loss function for training a model on a particular task/dataset is a particular objective function, and so the loss on many tasks corresponds to a set of objective functions. For clarity, we will refer to the algorithm that learns the optimization algorithm as the *meta-learner*, and the learned optimization algorithm as the *base-learner*.

First, let us consider what generalization means in this meta-learning setting. Akin to ordinary learning, the learned model should be evaluated on its ability to generalize to unseen data. Therefore, the learned optimization algorithm should be evaluated on its ability to generalize to unseen objective functions. Akin to the supervised learning paradigm, we divide the dataset of objective functions into training and test sets. At test time, the learned optimizer can be used stand-alone and functions exactly like a hand-engineered optimizer, except that the update formula is replaced with a neural net and no hyperparameters like step size or momentum need to be specified by the user. In particular, it should not perform multiple trials on the same objective function at test time, unlike hyperparameter optimization.

Just as in ordinary learning, it is important to identify what kinds of regularities or knowledge we can hope to learn and what we cannot under the setting of interest, in order to determine whether learning can be expected to yield anything useful. Then the natural question is: under this proposed setting of *learning how to learn*, what kinds of regularities can we expect to learn, and why are they useful? Let us consider a few different settings below.

2.2.1 Learning on One Objective Function

Suppose we only have a single objective function in the training set, and we aim to learn an optimization algorithm that converges quickly on this single objective function. What would the meta-learner learn in this case?

Consider an optimization algorithm that simply memorizes the optimum of the single objective function. This is the best possible optimization algorithm, since it always converges to the optimum in one step, regardless of initialization (as shown in Figure). So, at meta-training time, the meta-learner could simply find the optimum of the objective function using any off-the-shelf optimization algorithm and then simply store the optimum in the parameters of the base-learner.

While the meta-learner would be satisfied with such a solution, this is not very useful from a practical standpoint. Such a learned optimization algorithm does not generalize at all and cannot be used on any other objective function.

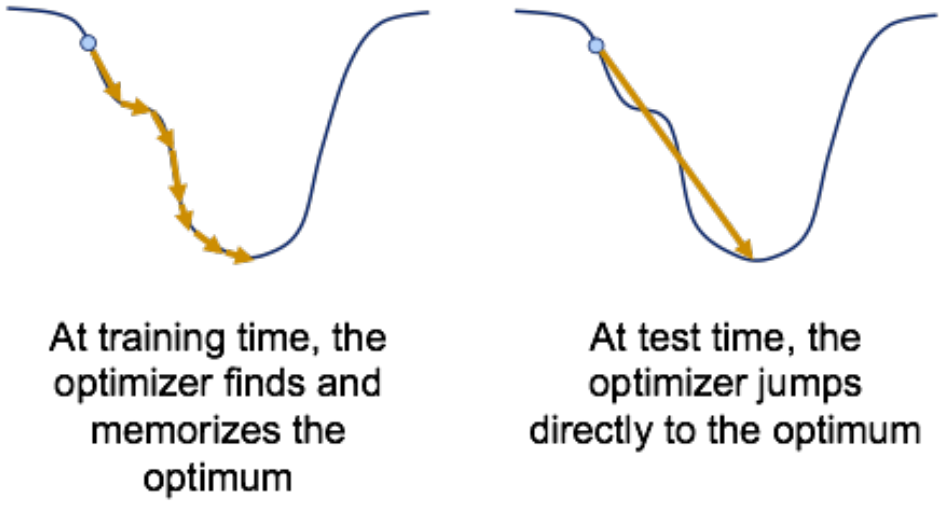


Figure 2.1: If we train the optimization on a single objective function, we can easily learn the location of the optimum of the objective function rather than a useful rule for optimizing it. In other words, the learned optimizer can simply memorize what the location of the optimum is.

2.2.2 Learning on Finitely Many Objective Functions

Now suppose we have a finitely many objective functions in the training set, and we aim to learn an optimization algorithm that converges quickly on this single objective function. What would the meta-learner learn in this case?

If the learned optimization algorithm could identify which objective function in the training set that it is optimizing, then it could immediately jump to the memorized optimum of that objective function. So, in other words, this setting would simply reduce to the previous

setting. Whether the meta-learner learns such an optimization algorithm depends on how hard it is to identify the objective function. So, in order the learned optimization algorithm to generalize, it must be hard to uniquely identify the objective function.

For a fixed set of n locations where the optimization algorithm evaluates an objective function, there must be at least 2^n objective functions to guarantee that the optimization algorithm cannot uniquely identify any objective function with less than n iterations.

So, if the learning formulation is not carefully designed, the meta-learner can easily learn a base-learner that fails to generalize to unseen objective functions.

To avoid this, there are two possibilities: (1) we can regularize the base-learner so that it can only remember information from a relatively small number of time steps, and/or (2) we can make the effective number of objective functions the meta-learner is trained on exponential in the number of time steps that can be remembered.

It turns out that both can be achieved with reinforcement learning: the former is achieved by having a relatively low-dimensional state space, and the latter is achieved by the Markovian structure of the dynamics/state transition probability.

2.2.3 Learning on All Possible Objective Functions

Now consider the other extreme of an infinitely large training set with all possible objective functions. Can the meta-learner learning anything useful in this case?

Consider an arbitrary optimization algorithm, which might perform quite well on most objective functions. Because it only relies on information at the previous iterates, we can modify the objective function at the last iterate to make it arbitrarily bad while maintaining the geometry of the objective function at all previous iterates. Then, on this modified objective function, by construction, the optimizer would follow the exact same trajectory as before and end up at a point with a bad objective value. Therefore, any optimizer has objective functions that it performs poorly on and no optimizer can perform well on all possible objective functions.

Hence, learning an optimization algorithm on all possible objective functions will not result in a useful optimizer, because there are no regularities that can be exploited by the learned optimization algorithm. As a result, meta-learning in this setting would not make sense.

2.2.4 When Does Meta-Learning Make Sense?

For meta-learning to be useful, there must be some underlying regularities across the subset of objective functions we might be interested in. These regularities can then be learned by the meta-learner and then exploited at meta-test time by the base-learner. These must be regularities that cannot be discovered by focusing on any particular objective function – if they were, then meta-learning would not offer any advantage over ordinary base-learning.

What could be an example of such regularities? One example is the shared geometry of a subset of objective functions we may encounter frequently in practice. More specifically,

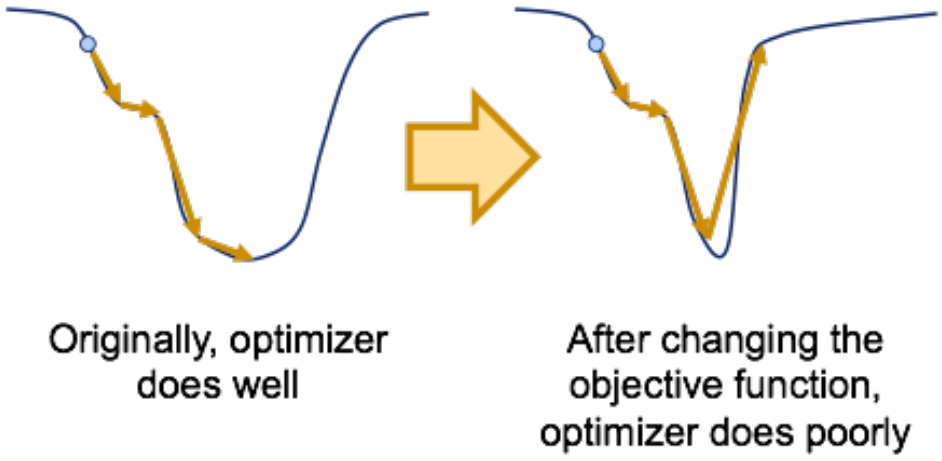


Figure 2.2: For any given optimizer, we can always construct an objective function on which it performs poorly. This implies that we cannot hope to learn an optimization algorithm that performs well on all possible objective functions.

consider the class of objective functions that represent a loss function composed with a neural net classifier of varying architectures with ReLU activations. There is regularity in this class objective functions, since the neural net associated with each objective function is guaranteed to be piecewise linear. In principle, the meta-learner could learn an optimization algorithm that can take advantage of this regularity.

2.2.5 Difference with Classical Meta-Learning

Most classical methods [120] for meta-learning perform what is now known as multi-task learning. In this setting, the goal is to learn commonalities across different tasks – for example, consider the setting with two tasks, one to localize, or find the location, of an object in an image, and one to classify the object into one of several different categories, a common preprocessing step that would be useful for both tasks is edge detection. A common example of a classical meta-learning approach is to divide the parameter space into two, one that is shared across the different tasks, and one that is specific to each task. For example, we can share the weights in the lower layers of a neural net across different tasks and have task-specific weights in the upper layers.

Our setting is different: our goal is learn not the commonalities that are shared across different tasks, but rather the commonalities shared by the *experience* of training on different tasks. The tasks themselves may be completely unrelated – for example, some could be tasks related to images, others could be tasks related to language. The only source of regularity could come from the geometry of the model class, or meta-properties of the dataset, like sparsity or degree of class imbalance.

2.3 Taxonomy of Meta-Learning

The terms "learning to learn" and "meta-learning" have appeared from time to time in the literature [13, 122, 32, 120] and refer to the general theme of learning meta-level knowledge that are useful across tasks. Despite the long history, the term "meta-learning" has been used by different authors to refer to disparate methods with different purposes; there is no consensus on what precisely meta-learning means and what differentiates meta-learning from ordinary (base-)learning. These methods all share the objective of learning some form of meta-knowledge about learning, but differ in the type of meta-knowledge they aim to learn. To improve the conceptual clarity of various meta-learning methods, we divide the various methods into the three categories: learning *what* to learn, learning *which model* to learn, and learning *how* to learn. The first two correspond to the classical work on meta-learning mentioned above; the last originated with the line of work on online hyperparameter adaptation and learning optimization algorithms.

2.3.1 Learning What to Learn

Methods in this category [120] aim to learn what parameter values of the base-level learner are useful across a family of related tasks. The meta-knowledge captures commonalities shared by tasks in the family, which enables learning on a new task from the family to be done more quickly. Most early methods fall into this category; this line of work has blossomed into an area that has later become known as transfer learning and multi-task learning.

2.3.2 Learning Which Model to Learn

Methods in this category [32] aim to learn which base-level learner achieves the best performance on a task. The meta-knowledge captures correlations between different tasks and the performance of different base-level learners on those tasks. One challenge under this setting is to decide on a parameterization of the space of base-level learners that is both rich enough to be capable of representing disparate base-level learners and compact enough to permit tractable search over this space. [31] proposes a nonparametric representation and stores examples of different base-level learners in a database, whereas [114] proposes representing base-level learners as general-purpose programs. The former has limited representation power, while the latter makes search and learning in the space of base-level learners intractable. [72] views the (online) training procedure of any base-learner as a black box function that maps a sequence of training examples to a sequence of predictions and models it as a recurrent neural net. Under this formulation, meta-training reduces to training the recurrent net, and the parameters of the base-level learner are entirely contained in the memory state of the recurrent net.

Hyperparameter optimization can be seen as another example of methods in this category. The space of base-level learners to search over is parameterized by a predefined set of hyperparameters. Unlike the methods above, multiple trials with different hyperparameter

settings on the same task are permitted, and so generalization across tasks is not required. The discovered hyperparameters are generally specific to the task at hand and hyperparameter optimization must be rerun for new tasks. Various kinds of methods have been proposed, such those based on Bayesian optimization [74, 24, 117, 119, 52], random search [25] and gradient-based optimization [17, 44, 97].

2.3.3 Learning How to Learn

Methods in this category aim to learn a good algorithm for training a base-level learner. Unlike methods in the previous categories, the goal is not to learn about the *outcome* of learning, but rather the *process* of learning. The meta-knowledge captures commonalities in the behaviours of learning algorithms that achieve good performance. The base-level learner and the task are given by the user, so the learned algorithm must generalize across base-level learners and tasks. Since learning in most cases is equivalent to optimizing some objective function, learning a learning algorithm often reduces to learning an optimization algorithm.

The method presented in this dissertation was the first to learn a general-purpose optimization algorithm; by general-purpose, we mean two attributes: the meta-learner learns both the step size and step direction with which to update the parameters of the base-learner, and the same learned optimization algorithm can be on different objective functions. Concurrently, [4] explores a similar theme under a different setting, where the goal is learn a task-dependent optimization algorithm. The optimizer is trained from the experience of training on a particular task or family of tasks and is evaluated on its ability to train on the same task or family of tasks. Under this setting, the optimizer learns regularities about the task itself rather than regularities of the model in general. As a result, it is unable to generalize to unseen tasks/objective functions. Closely related is [16], which learns a Hebb-like synaptic learning rule that does not depend on the objective function, which does not allow for generalization to different objective functions.

Various work has explored learning how to adjust the hyperparameters of hand-engineered optimization algorithms, like the step size [66, 39, 56] or the damping factor in the Levenberg-Marquardt algorithm [111]. Related to this line of work is stochastic meta-descent [30], which derives a rule for adjusting the step size analytically. A different line of work [59, 118] parameterizes intermediate operands of special-purpose solvers for a class of optimization problems that arise in sparse coding and learns them using supervised learning.

Because methods in this category learn an algorithm, it is related to program induction, which considers the problem of learning programs from examples of input and output. Several different approaches have been proposed: genetic programming [38] represents programs as abstract syntax trees and evolves them using genetic algorithms, [94] represents programs explicitly using a formal language, constructs a hierarchical Bayesian prior over programs and performs inference using an MCMC sampling procedure and [58] represents programs implicitly as sequences of memory access operations and trains a recurrent neural net to learn the underlying patterns in the memory access operations. [72] considers the special case of online learning algorithms, each of which is represented as a recurrent neural net

with a particular setting of weights, and learns the online learning algorithm by learning the neural net weights. While the program/algorithm improves as training progresses, the algorithms learned using these methods have not been able to match the performance of simple hand-engineered algorithms. In contrast, our aim is learn an algorithm that is better than known hand-engineered algorithms.

2.4 How to Learn the Optimizer

The first approach we tried was to treat the problem of learning optimizers as a standard supervised learning problem: we simply differentiate the meta-loss with respect to the parameters of the update formula and learn these parameters using standard gradient-based optimization.

This seemed like a natural approach, but it did not work: despite our best efforts, we could not get any optimizer trained in this manner to generalize to unseen objective functions, even though they were drawn from the same distribution that generated the objective functions used to train the optimizer. On almost all unseen objective functions, the learned optimizer started off reasonably, but quickly diverged after a while. On the other hand, on the training objective functions, it exhibited no such issues and did quite well. Why is this?

It turns out that optimizer learning is not as simple a learning problem as it appears. Standard supervised learning assumes all training examples are independent and identically distributed (i.i.d.); in our setting, the step vector the optimizer takes at any iteration affects the gradients it sees at all subsequent iterations. Furthermore, how the step vector affects the gradient at the subsequent iteration is not known, since this depends on the local geometry of the objective function, which is unknown at meta-test time. Supervised learning cannot operate in this setting, and must assume that the local geometry of an unseen objective function is the same as the local geometry of training objective functions at all iterations.

Consider what happens when an optimizer trained using supervised learning is used on an unseen objective function. It takes a step, and discovers at the next iteration that the gradient is different from what it expected. It then recalls what it did on the training objective functions when it encountered such a gradient, which could have happened in a completely different region of the space, and takes a step accordingly. To its dismay, it finds out that the gradient at the next iteration is even more different from what it expected. This cycle repeats and the error the optimizer makes becomes bigger and bigger over time, leading to rapid divergence.

This phenomenon is known in the literature as the problem of compounding errors. It is known that the total error of a supervised learner scales quadratically in the number of iterations, rather than linearly as would be the case in the i.i.d. setting **??**. In essence, an optimizer trained using supervised learning necessarily overfits to the geometry of the training objective functions. One way to solve this problem is to use reinforcement learning.

More specifically, through the lens of reinforcement learning, a particular optimization algorithm simply corresponds to a policy. Learning an optimization algorithm then reduces

to finding an optimal policy. For this purpose, we use an off-the-shelf reinforcement learning algorithm known as guided policy search [86], which has demonstrated success in a variety of robotic control settings [88, 53, 87, 65].

2.5 Reinforcement Learning

2.5.1 Markov Decision Process

In the reinforcement learning setting, the learner is given a choice of actions to take in each time step, which changes the state of the environment in an unknown fashion, and receives feedback based on the consequence of the action. The feedback is typically given in the form of a reward or cost, and the objective of the learner is to choose a sequence of actions based on observations of the current environment that maximizes cumulative reward or minimizes cumulative cost over all time steps.

More formally, a reinforcement learning problem can be characterized by a Markov decision process (MDP). We consider an undiscounted finite-horizon MDP with continuous state and action spaces defined by the tuple $(\mathcal{S}, \mathcal{A}, p_0, p, c, T)$, where $\mathcal{S} \subseteq \mathbb{R}^D$ is the set of states, $\mathcal{A} \subseteq \mathbb{R}^d$ is the set of actions, $p_0 : \mathcal{S} \to \mathbb{R}^+$ is the probability density over initial states, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^+$ is the transition probability density, that is, the conditional probability density over successor states given the current state and action, $c : \mathcal{S} \to \mathbb{R}$ is a function that maps state to cost and T is the time horizon. A policy $\pi : \mathcal{S} \times \mathcal{A} \times \{0, \ldots, T-1\} \to \mathbb{R}^+$ is a conditional probability density over actions given the state at each time step. When a policy is independent of the time step, it is referred to as stationary.

2.5.2 Policy Search

This problem of finding the cost-minimizing policy is known as the policy search problem. More precisely, the objective is to find a policy π^* such that

$$\pi^* = \arg\min_{\pi} \mathbb{E}_{s_0, a_0, s_1, \ldots, s_T} \left[\sum_{t=0}^{T} c(s_t) \right],$$

where the expectation is taken with respect to the joint distribution over the sequence of states and actions, often referred to as a trajectory, which has the density

$$q\left(s_0, a_0, s_1, \ldots, s_T\right) = p_0\left(s_0\right) \prod_{t=0}^{T-1} \pi\left(a_t \middle| s_t, t\right) p\left(s_{t+1} \middle| s_t, a_t\right).$$

To enable generalization to unseen states, the policy is typically parameterized and minimization is performed over representable policies. Solving this problem exactly is intractable in all but selected special cases. Therefore, policy search methods generally tackle this problem by solving it approximately. In addition, the transition probability density p is typically not known, but may be accessed via sampling.

2.5.3 Guided Policy Search

Guided policy search (GPS) [86] is a method for searching over expressive non-linear policy classes in continuous state and action spaces. It works by alternating between computing a mixture of target trajectories and training the policy to replicate them. Successive iterations locally improve target trajectories while ensuring proximity to behaviours that are reproducible by the policy. Target trajectories are computed by fitting local approximations to the cost and transition probability density and optimizing over a restricted class of time-varying linear target policies subject to a trust region constraint. The stationary non-linear policy is trained to minimize the squared Mahalanobis distance between the predicted and target actions at each time step.

More precisely, GPS works by solving the following constrained optimization problem:

$$\min_{\theta,\eta} \mathbb{E}_{\psi}\left[\sum_{t=0}^{T} c(s_t)\right] \text{ s.t. } \psi\left(a_t|\, s_t, t; \eta\right) = \pi\left(a_t|\, s_t; \theta\right) \ \forall a_t, s_t, t,$$

where ψ denotes the time-varying target policy, π denotes the stationary non-linear policy, and $\mathbb{E}_{\psi}[\cdot]$ denotes the expectation taken with respect to the trajectory induced by the target policy ψ. ψ is assumed to be conditionally Gaussian whose mean is linear in s_t and π is assumed to be conditionally Gaussian whose mean could be an arbitrary function of s_t. To solve this problem, the equality constraint is relaxed and replaced with a penalty on the KL-divergence between ψ and π. Different flavours of GPS [86, 88] use different constrained optimization methods, which all involve alternating between optimizing the parameters of ψ and π.

For updating ψ, GPS first builds a model $\tilde{p}$ of the transition probability density p of the form $\tilde{p}\left(s_{t+1}|\, s_t, a_t, t\right) \coloneqq \mathcal{N}(A_t s_t + B_t a_t + c_t, F_t)$[1], where A_t, B_t and c_t are parameters estimated from samples drawn from the trajectory induced by the existing ψ. It also computes local quadratic approximations to the cost, so that $c(s_t) \approx \frac{1}{2} s_t^T C_t s_t + d_t^T s_t + h_t$ for s_t's that are near the samples. It then solves the following:

$$\min_{K_t, k_t, G_t} \mathbb{E}_{\tilde{\psi}}\left[\sum_{t=0}^{T} \frac{1}{2} s_t^T C_t s_t + d_t^T s_t\right]$$
$$\text{s.t. } \sum_{t=0}^{T} D_{KL}\left(p\left(s_t\right)\psi\left(\cdot|\, s_t, t; \eta\right) \| \, p\left(s_t\right)\psi\left(\cdot|\, s_t, t; \eta'\right)\right) \leq \epsilon,$$

where $\mathbb{E}_{\tilde{\psi}}[\cdot]$ denotes the expectation taken with respect to the trajectory induced by the target policy ψ if states transition according to the model $\tilde{p}$. K_t, k_t, G_t are the parameters of $\psi\left(a_t|\, s_t, t; \eta\right) \coloneqq \mathcal{N}\left(K_t s_t + k_t, G_t\right)$ and η' denotes the parameters of the previous target policy. It turns out that this optimization problem can be solved in closed form using a dynamic programming algorithm known as linear-quadratic-Gaussian regulator (LQG).

[1] In a slight abuse of notation, we use $\mathcal{N}(\mu, \Sigma)$ to denote the density of a Gaussian distribution with mean μ and covariance Σ.

For updating π, GPS minimizes $D_{KL}\left(p\left(s_t\right)\pi\left(\cdot|\, s_t\right)\|\, p\left(s_t\right)\psi\left(\cdot|\, s_t, t\right)\right)$. Assuming fixed covariance and omitting dual variables, this corresponds to minimizing the following:

$$\mathbb{E}_{\psi}\left[\sum_{t=0}^{T}\left(\mathbb{E}_{\pi}\left[a_t|\, s_t\right]-\mathbb{E}_{\psi}\left[a_t|\, s_t, t\right]\right)^T G_t^{-1}\left(\mathbb{E}_{\pi}\left[a_t|\, s_t\right]-\mathbb{E}_{\psi}\left[a_t|\, s_t, t\right]\right)\right],$$

where $\mathbb{E}_{\pi}\left[\cdot\right]$ denotes the expectation taken with respect to the trajectory induced by the non-linear policy π. We refer interested readers to [86] and [88] for details.

2.6 Formulation

We observe that the execution of an optimization algorithm can be viewed as the execution of a particular policy in an MDP: the state consists of the current iterate and the objective values and gradients evaluated at the current and past iterates, the action is the step vector that is used to update the current iterate, and the transition probability is partially characterized by the update formula, $x^{(i)} \leftarrow x^{(i-1)} + \Delta x$. The policy that is executed corresponds precisely to the choice of π used by the optimization algorithm. For this reason, we will also use π to denote the policy at hand. Under this formulation, searching over policies corresponds to searching over possible optimization algorithms.

To learn π, we need to define the cost function, which should penalize policies that exhibit undesirable behaviours during their execution. Since the performance metric of interest for optimization algorithms is the speed of convergence, the cost function should penalize policies that converge slowly. To this end, assuming the goal is to minimize the objective function, we define cost at a state to be the objective value at the current iterate. This encourages the policy to reach the minimum of the objective function as quickly as possible. We choose to parameterize the mean of π using a neural net, due to its appealing properties as a universal function approximator and strong empirical performance in a variety of applications. We use GPS to learn π.

2.7 Implementation Details

We store the current iterate, previous gradients and improvements in the objective value from previous iterations in the state. We keep track of only the information pertaining to the previous H time steps and use $H = 25$ in our experiments. More specifically, the dimensions of the state space encode the following information:

- Current iterate
- Change in the objective value at the current iterate relative to the objective value at the i^{th} most recent iterate for all $i \in \{2, \ldots, H+1\}$

- Gradient of the objective function evaluated at the i^{th} most recent iterate for all $i \in \{2, \ldots, H+1\}$

Initially, we set the dimensions corresponding to historical information to zero. The current iterate is only used to compute the cost; because the policy should not depend on the absolute coordinates of the current iterate, we exclude it from the input that is fed into the neural net.

We use a small neural net with a single hidden layer of 50 hidden units to model the mean of π. Softplus activation units are used at the hidden layer and linear activation units are used at the output layer. We initialize the weights of the neural net randomly and do not regularize the magnitude of weights.

Initially, we set the target trajectory distribution so that the mean action given state at each time step matches the step vector used by the gradient descent method with momentum. We choose the best settings of the step size and momentum decay factor for each objective function in the training set by performing a grid search over hyperparameters and running noiseless gradient descent with momentum for each hyperparameter setting. We use a mixture of 10 Gaussians as a prior for fitting the parameters of the transition probability density.

For training, we sample 20 trajectories with a length of 40 time steps for each objective function in the training set. After each iteration of guided policy search, we sample new trajectories from the new distribution and discard the trajectories from the preceding iteration.

2.8 Experiments

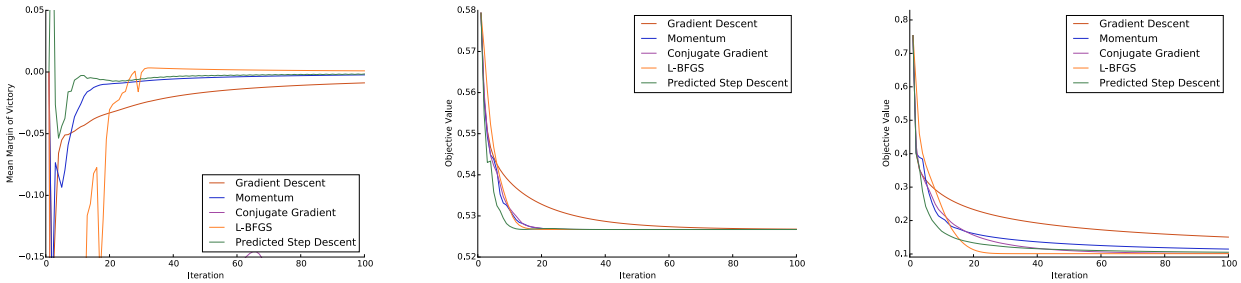
We learn optimization algorithms for various convex and non-convex classes of objective functions that correspond to loss functions for different machine learning models. We learn an optimizer for logistic regression, robust linear regression using the Geman-McClure M-estimator and a two-layer neural net classifier with ReLU activation units. The geometry of the error surface becomes progressively more complex: the loss for logistic regression is convex, the loss for robust linear regression is non-convex, and the loss for the neural net has many local minima.

2.8.1 Logistic Regression

We consider a logistic regression model with an ℓ_2 regularizer on the weight vector. Training the model requires optimizing the following objective:

$$\min_{\mathbf{w},b} -\frac{1}{n}\sum_{i=1}^{n} y_i \log \sigma\left(\mathbf{w}^T\mathbf{x}_i + b\right) + (1-y_i)\log\left(1-\sigma\left(\mathbf{w}^T\mathbf{x}_i+b\right)\right) + \frac{\lambda}{2}\left\|\mathbf{w}\right\|_2^2,$$

where $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ denote the weight vector and bias respectively, $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$ denote the feature vector and label of the i^{th} instance, λ denotes the coefficient



(a) (b) (c)

Figure 2.3: (a) Mean margin of victory of each algorithm for optimizing the logistic regression loss. Higher margin of victory indicates better performance. (b-c) Objective values achieved by each algorithm on two objective functions from the test set. Lower objective values indicate better performance. Best viewed in colour.

on the regularizer and $\sigma(z) := \frac{1}{1+e^{-z}}$. For our experiments, we choose $\lambda = 0.0005$ and $d = 3$. This objective is convex in $\mathbf{w}$ and b.

We train an algorithm for optimizing objectives of this form. Different examples in the training set correspond to such objective functions with different instantiations of the free variables, which in this case are $\mathbf{x}_i$ and y_i. Hence, each objective function in the training set corresponds to a logistic regression problem on a different dataset.

To construct the training set, we randomly generate a dataset of 100 instances for each function in the training set. The instances are drawn randomly from two multivariate Gaussians with random means and covariances, with half drawn from each. Instances from the same Gaussian are assigned the same label and instances from different Gaussians are assigned different labels.

We train the optimizer on a set of 90 objective functions. We evaluate it on a test set of 100 random objective functions generated using the same procedure and compare to popular hand-engineered algorithms, such as gradient descent, momentum, conjugate gradient and L-BFGS. All baselines are run with the best hyperparameter settings tuned on the training set.

For each algorithm and objective function in the test set, we compute the difference between the objective value achieved by a given algorithm and that achieved by the best of the competing algorithms at every iteration, a quantity we will refer to as "the margin of victory". This quantity is positive when the current algorithm is better than all other algorithms and negative otherwise. In Figure 2.3a, we plot the mean margin of victory of each algorithm at each iteration averaged over all objective functions in the test set.

As shown, the learned optimizer, which we will henceforth refer to as "predicted step descent", outperforms gradient descent, momentum and conjugate gradient at almost every iteration. The margin of victory for predicted step descent is high in early iterations, indicating that it converges much faster than other algorithms. It is interesting to note that despite

having seen only trajectories of length 40 at training time, the learned optimizer is able to generalize to much longer time horizons at test time. L-BFGS converges to slightly better optima than predicted step descent and the momentum method. This is not surprising, as the objective functions are convex and L-BFGS is known to be a very good optimizer for convex problems.

We show the performance of each algorithm on two objective functions from the test set in Figures 2.3b and 2.3c. In Figure 2.3b, predicted step descent converges faster than all other algorithms. In Figure 2.3c, predicted step descent initially converges faster than all other algorithms but is later overtaken by L-BFGS, while remaining faster than all other optimizers. However, it eventually achieves the same objective value as L-BFGS, while the objective values achieved by gradient descent and momentum remain much higher.

2.8.2 Robust Linear Regression

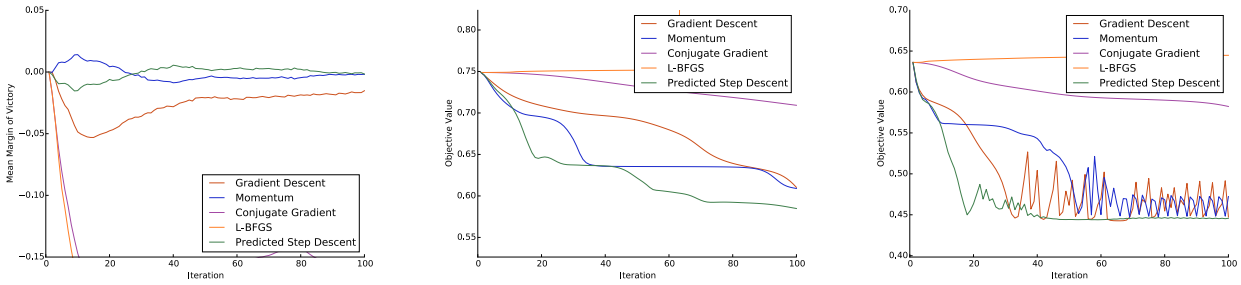
Next, we consider the problem of linear regression using a robust loss function. One way to ensure robustness is to use an M-estimator for parameter estimation. A popular choice is the Geman-McClure estimator, which induces the following objective:

$$\min_{\mathbf{w},b} \frac{1}{n} \sum_{i=1}^{n} \frac{\left(y_i - \mathbf{w}^T \mathbf{x}_i - b\right)^2}{c^2 + \left(y_i - \mathbf{w}^T \mathbf{x}_i - b\right)^2},$$

where $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ denote the weight vector and bias respectively, $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$ denote the feature vector and label of the i^{th} instance and $c \in \mathbb{R}$ is a constant that modulates the shape of the loss function. For our experiments, we use $c = 1$ and $d = 3$. This loss function is not convex in either $\mathbf{w}$ or b.

As with the preceding section, each objective function in the training set is a function of the above form with a particular instantiation of $\mathbf{x}_i$ and y_i. The dataset for each objective function is generated by drawing 25 random samples from each one of four multivariate Gaussians, each of which has a random mean and the identity covariance matrix. For all points drawn from the same Gaussian, their labels are generated by projecting them along the same random vector, adding the same randomly generated bias and perturbing them with i.i.d. Gaussian noise.

The optimizer is trained on a set of 120 objective functions. We evaluate it on 100 randomly generated objective functions using the same metric as above. As shown in Figure 2.4a, predicted step descent outperforms all hand-engineered algorithms except at early iterations. While it dominates gradient descent, conjugate gradient and L-BFGS at all times, it does not make progress as quickly as the momentum method initially. However, after around 30 iterations, it is able to close the gap and surpass the momentum method. On this optimization problem, both conjugate gradient and L-BFGS diverge quickly. Interestingly, unlike in the previous experiment, L-BFGS no longer performs well, which could be caused by non-convexity of the objective functions.



(a) (b) (c)

Figure 2.4: (a) Mean margin of victory of each algorithm for optimizing the robust linear regression loss. Higher margin of victory indicates better performance. (b-c) Objective values achieved by each algorithm on two objective functions from the test set. Lower objective values indicate better performance. Best viewed in colour.

Figures 2.4b and 2.4c show performance on objective functions from the test set. In Figure 2.4b, predicted step descent not only converges the fastest, but also reaches a better optimum than all other algorithms. In Figure 2.4c, predicted step descent converges the fastest and is able to avoid most of the oscillations that hamper gradient descent and momentum after reaching the optimum.

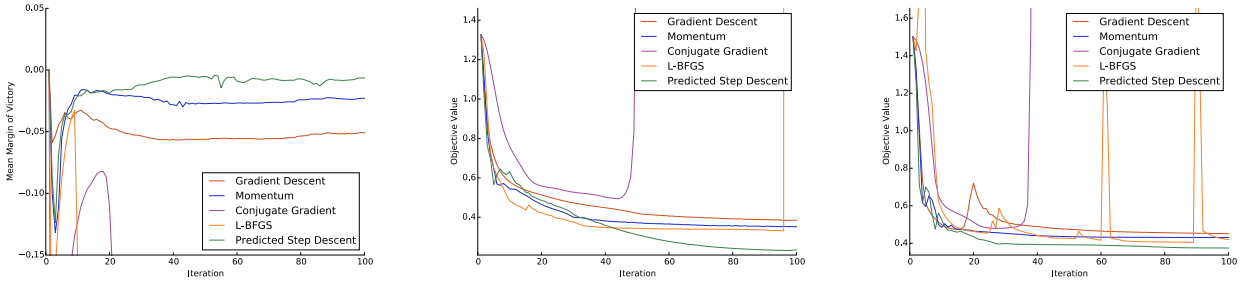
2.8.3 Neural Net Classifier

Finally, we train an optimizer to train a small neural net classifier. We consider a two-layer neural net with ReLU activation on the hidden units and softmax activation on the output units. We use the cross-entropy loss combined with ℓ_2 regularization on the weights. To train the model, we need to optimize the following objective:

$$\min_{W,U,\mathbf{b},\mathbf{c}} -\frac{1}{n}\sum_{i=1}^{n}\log\left(\frac{\exp\left((U\max\left(W\mathbf{x}_i+\mathbf{b},0\right)+\mathbf{c})_{y_i}\right)}{\sum_j \exp\left((U\max\left(W\mathbf{x}_i+\mathbf{b},0\right)+\mathbf{c})_j\right)}\right)+\frac{\lambda}{2}\left\|W\right\|_F^2+\frac{\lambda}{2}\left\|U\right\|_F^2,$$

where $W \in \mathbb{R}^{h\times d}, b \in \mathbb{R}^h, U \in \mathbb{R}^{p\times h}, c \in \mathbb{R}^p$ denote the first-layer and second-layer weights and biases, $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{1,\ldots,p\}$ denote the input and target class label of the i^{th} instance, λ denotes the coefficient on regularizers and $(\mathbf{v})_j$ denotes the j^{th} component of $\mathbf{v}$. For our experiments, we use $\lambda = 0.0005$ and $d = h = p = 2$. The error surface is known to have complex geometry and multiple local optima, making this a challenging optimization problem.

The training set consists of 80 objective functions, each of which corresponds to the objective for training a neural net on a different dataset. Each dataset is generated by generating four multivariate Gaussians with random means and covariances and sampling



(a) (b) (c)

Figure 2.5: (a) Mean margin of victory of each algorithm for training neural net classifiers. Higher margin of victory indicates better performance. (b-c) Objective values achieved by each algorithm on two objective functions from the test set. Lower objective values indicate better performance. Best viewed in colour.

25 points from each. The points from the same Gaussian are assigned the same random label of either 0 or 1. We make sure not all of the points in the dataset are assigned the same label.

We evaluate the learned optimizer in the same manner as above. As shown in Figure 2.5a, predicted step descent significantly outperforms all other algorithms. In particular, as evidenced by the sizeable and sustained gap between margin of victory for predicted step descent and the momentum method, predicted step descent is able to reach much better optima and is less prone to getting trapped in local optima compared to other methods. This gap is also larger compared to that exhibited in previous sections, suggesting that hand-engineered algorithms are more sub-optimal on challenging optimization problems and so the potential for improvement from learning the algorithm is greater in such settings. Due to non-convexity, conjugate gradient and L-BFGS often diverge.

Performance on examples of objective functions from the test set is shown in Figures 2.5b and 2.5c. As shown, predicted step descent is able to reach better optima than all other methods and largely avoids oscillations that other methods suffer from.

2.8.4 Visualization of Optimization Trajectories

We visualize optimization trajectories followed by the learned algorithm and various hand-engineered algorithms to gain further insights into the behaviour of the learned algorithm. We generated random two-dimensional logistic regression problems and plot trajectories followed by different algorithms on each problem in Figure 2.6.

As shown, the learned algorithm exhibits some interesting behaviours. In Figure 2.6a, the learned algorithm does not take as large a step as L-BFGS initially, but takes larger steps than L-BFGS later on as it approaches the optimum. In other words, the learned algorithm appears to be not as greedy as L-BFGS. In Figures 2.6b and 2.6d, the learned

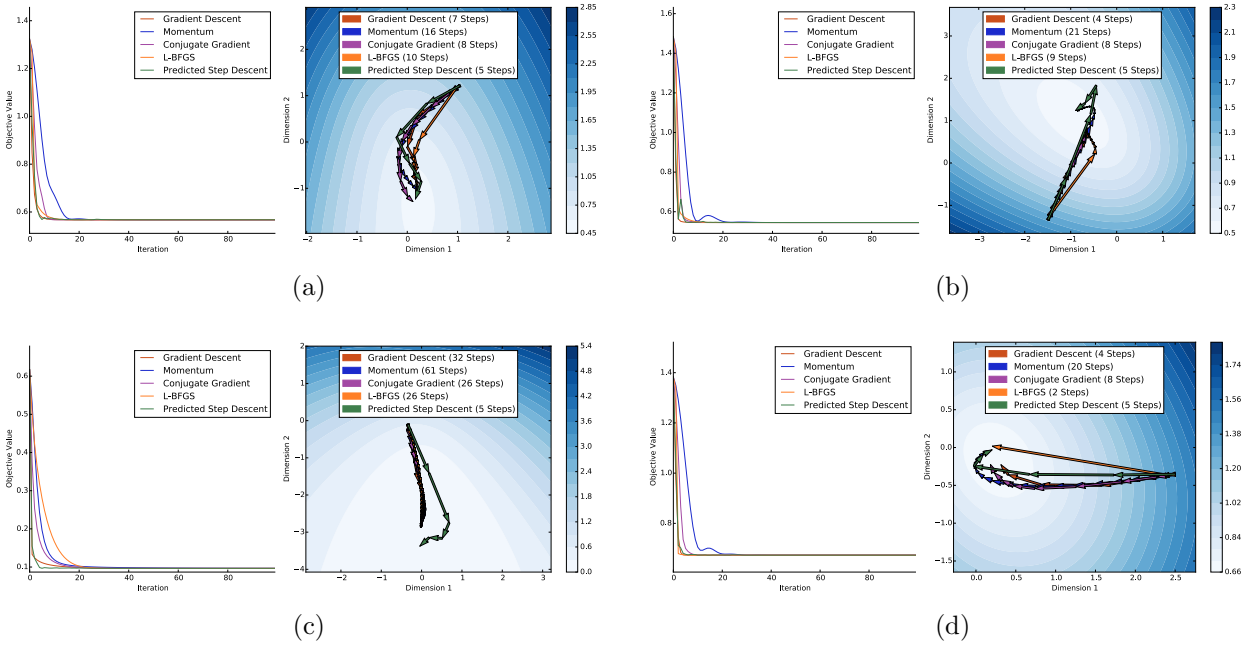
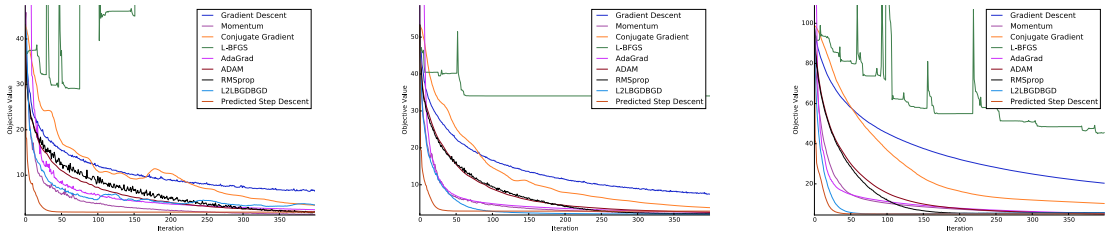


Figure 2.6: Objective values and trajectories produced by different algorithms on unseen random two-dimensional logistic regression problems. Each pair of plots corresponds to a different logistic regression problem. Objective values are shown on the vertical axis in the left plot and as contour levels in the right plot, where darker shading represents higher objective values. In the right plot, the axes represent the values of the iterates in each dimension and are of the same scale. Each arrow represents one iteration of an algorithm, whose tail and tip correspond to the preceding and subsequent iterates respectively. Best viewed in colour.

algorithm initially overshoots, but appears to have learned how to recover while avoiding oscillations. In Figure 2.6c, the learned algorithm is able to make rapid progress despite vanishing gradients.

2.9 Learning Optimizers for High-Dimensional Problems

The problem of learning high-dimensional optimization algorithms presents challenges for reinforcement learning algorithms due to high dimensionality of the state and action spaces. For example, in the case of GPS, because the running time of LQG is cubic in dimensionality of the state space, performing policy search even in the simple class of linear-Gaussian policies would be prohibitively expensive when the dimensionality of the optimization problem is high.



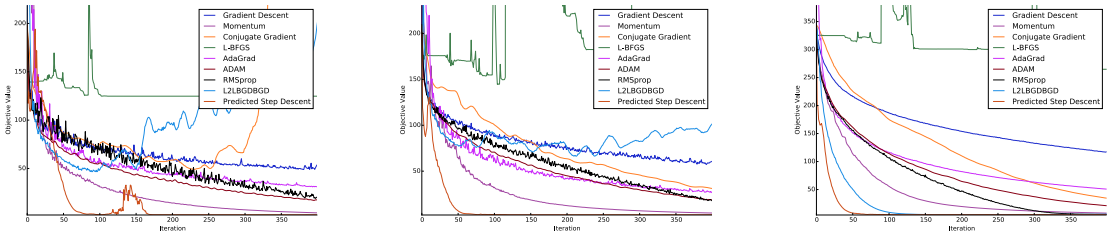
(a) (b) (c)

Figure 2.7: Comparison of the various hand-engineered and learned algorithms on training neural nets with 48 input and hidden units on (a) TFD, (b) CIFAR-10 and (c) CIFAR-100 with mini-batches of size 64. The vertical axis is the true objective value and the horizontal axis represents the iteration. Best viewed in colour.

Fortunately, many high-dimensional optimization problems have underlying structure that can be exploited. For example, the parameters of neural nets are equivalent up to permutation among certain coordinates. More concretely, for fully connected neural nets, the dimensions of a hidden layer and the corresponding weights can be permuted arbitrarily without changing the function they compute. Because permuting the dimensions of two adjacent layers can permute the weight matrix arbitrarily, an optimization algorithm should be invariant to permutations of the rows and columns of a weight matrix. A reasonable prior to impose is that the algorithm should behave in the same manner on all coordinates that correspond to entries in the same matrix. That is, if the values of two coordinates in all current and past gradients and iterates are identical, then the step vector produced by the algorithm should have identical values in these two coordinates. We will refer to the set of coordinates on which permutation invariance is enforced as a coordinate group. For the purposes of learning an optimization algorithm for neural nets, a natural choice would be to make each coordinate group correspond to a weight matrix or a bias vector. Hence, the total number of coordinate groups is twice the number of layers, which is usually fairly small.

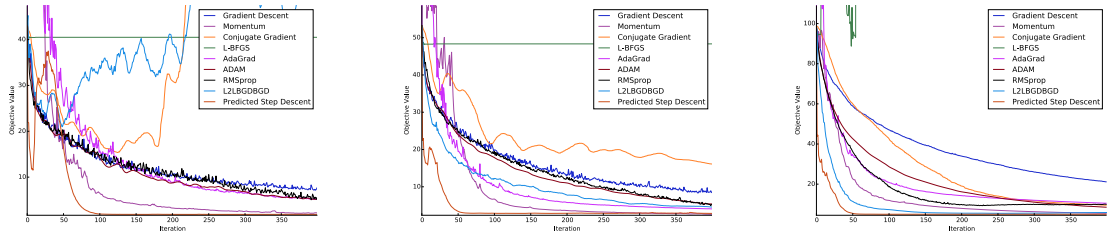
In the case of GPS, we impose this prior on both ψ and π. For the purposes of updating η, we first impose a block-diagonal structure on the parameters A_t, B_t and F_t of the fitted transition probability density $\tilde{p}\left(s_{t+1} \middle| s_t, a_t, t; \zeta\right) = \mathcal{N}(A_t s_t + B_t a_t + c_t, F_t)$, so that for each coordinate in the optimization problem, the dimensions of s_{t+1} that correspond to the coordinate only depend on the dimensions of s_t and a_t that correspond to the same coordinate. As a result, $\tilde{p}\left(s_{t+1} \middle| s_t, a_t, t; \zeta\right)$ decomposes into multiple independent probability densities $\tilde{p}^j\left(s_{t+1}^j \middle| s_t^j, a_t^j, t; \zeta^j\right)$, one for each coordinate j. Similarly, we also impose a block-diagonal structure on C_t for fitting $\tilde{c}(s_t)$ and on the parameter matrix of the fitted model for $\pi\left(a_t \middle| s_t; \theta\right)$. Under these assumptions, K_t and G_t are guaranteed to be block-diagonal as well. Hence, the Bregman divergence penalty term, $D\left(\eta, \theta\right)$ decomposes into a sum of Bregman divergence terms, one for each coordinate.

We then further constrain dual variables λ_t, sub-vectors of parameter vectors and sub-



(a) (b) (c)

Figure 2.8: Comparison of the various hand-engineered and learned algorithms on training neural nets with 100 input units and 200 hidden units on (a) TFD, (b) CIFAR-10 and (c) CIFAR-100 with mini-batches of size 64. The vertical axis is the true objective value and the horizontal axis represents the iteration. Best viewed in colour.



(a) (b) (c)

Figure 2.9: Comparison of the various hand-engineered and learned algorithms on training neural nets with 48 input and hidden units on (a) TFD, (b) CIFAR-10 and (c) CIFAR-100 with mini-batches of size 10. The vertical axis is the true objective value and the horizontal axis represents the iteration. Best viewed in colour.

matrices of parameter matrices corresponding to each coordinate group to be identical across the group. Additionally, we replace the weight ν_t on $D(\eta, \theta)$ with an individual weight on each Bregman divergence term for each coordinate group. The problem then decomposes into multiple independent subproblems, one for each coordinate group. Because the dimensionality of the state subspace corresponding to each coordinate is constant, LQG can be executed on each subproblem much more efficiently.

Similarly, for π, we choose a $\mu^{\pi}_{\omega}(\cdot)$ that shares parameters across different coordinates in the same group. We also impose a block-diagonal structure on Σ^{π} and constrain the appropriate sub-matrices to share their entries.

2.9.1 Features

We describe the features $\Phi(\cdot)$ and $\Psi(\cdot)$ at time step t, which define the state s_t and observation o_t respectively.

Because of the stochasticity of gradients and objective values, the state features $\Phi(\cdot)$ are defined in terms of summary statistics of the history of iterates $\left\{x^{(i)}\right\}_{i=0}^{t}$, gradients $\left\{\nabla \hat{f}(x^{(i)})\right\}_{i=0}^{t}$ and objective values $\left\{\hat{f}(x^{(i)})\right\}_{i=0}^{t}$. We define the following statistics, which we will refer to as the average recent iterate, gradient and objective value respectively:

- $\overline{x^{(i)}} \coloneqq \frac{1}{\min(i+1,3)} \sum_{j=\max(i-2,0)}^{i} x^{(j)}$
- $\overline{\nabla \hat{f}(x^{(i)})} \coloneqq \frac{1}{\min(i+1,3)} \sum_{j=\max(i-2,0)}^{i} \nabla \hat{f}(x^{(j)})$
- $\overline{\hat{f}(x^{(i)})} \coloneqq \frac{1}{\min(i+1,3)} \sum_{j=\max(i-2,0)}^{i} \hat{f}(x^{(j)})$

The state features $\Phi(\cdot)$ consist of the relative change in the average recent objective value, the average recent gradient normalized by the magnitude of the a previous average recent gradient and a previous change in average recent iterate relative to the current change in average recent iterate:

- $\left\{\left(\overline{\hat{f}(x^{(t-5i)})} - \overline{\hat{f}(x^{(t-5(i+1))})}\right) / \overline{\hat{f}(x^{(t-5(i+1))})}\right\}_{i=0}^{24}$
- $\left\{\overline{\nabla \hat{f}(x^{(t-5i)})} / \left(\left|\overline{\nabla \hat{f}(x^{(\max(t-5(i+1),t \bmod 5))})}\right| + 1\right)\right\}_{i=0}^{25}$
- $\left\{\frac{\left|\overline{x^{(\max(t-5(i+1),t \bmod 5+5))}} - \overline{x^{(\max(t-5(i+2),t \bmod 5))}}\right|}{\left|\overline{x^{(t-5i)}} - \overline{x^{(t-5(i+1))}}\right| + 0.1}\right\}_{i=0}^{24}$

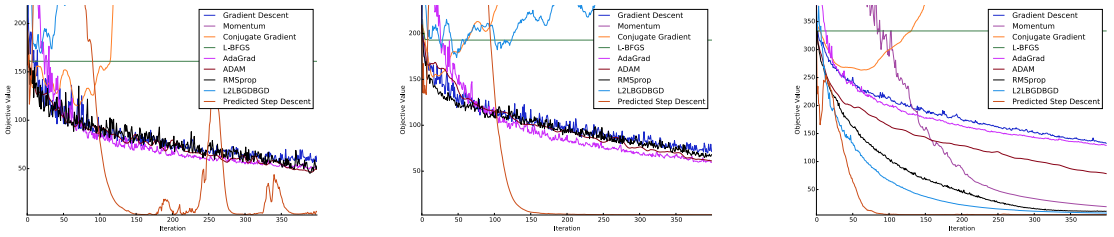
Note that all operations are applied element-wise. Also, whenever a feature becomes undefined (i.e.: when the time step index becomes negative), it is replaced with the all-zeros vector.

Unlike state features, which are only used when training the optimization algorithm, observation features $\Psi(\cdot)$ are used both during training and at test time. Consequently, we use noisier observation features that can be computed more efficiently and require less memory overhead. The observation features consist of the following:

- $\left(\hat{f}(x^{(t)}) - \hat{f}(x^{(t-1)})\right) / \hat{f}(x^{(t-1)})$
- $\nabla \hat{f}(x^{(t)}) / \left(\left|\nabla \hat{f}(x^{(\max(t-1,0))})\right| + 1\right)$
- $\frac{\left|x^{(\max(t-1,1))} - x^{(\max(t-2,0))}\right|}{\left|x^{(t)} - x^{(t-1)}\right| + 0.1}$

2.9.2 Experiments

For clarity, we will refer to training of the optimization algorithm as "meta-training" to differentiate it from base-level training, which will simply be referred to as "training".



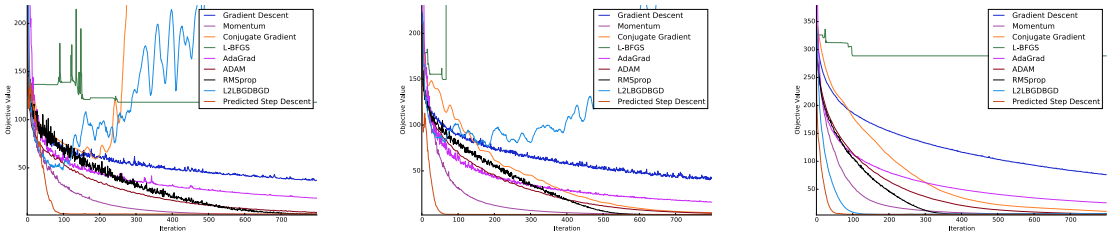
(a) (b) (c)

Figure 2.10: Comparison of the various hand-engineered and learned algorithms on training neural nets with 100 input units and 200 hidden units on (a) TFD, (b) CIFAR-10 and (c) CIFAR-100 with mini-batches of size 10. The vertical axis is the true objective value and the horizontal axis represents the iteration. Best viewed in colour.

We meta-trained an optimization algorithm on a single objective function, which corresponds to the problem of training a two-layer neural net with 48 input units, 48 hidden units and 10 output units on a randomly projected and normalized version of the MNIST training set with dimensionality 48 and unit variance in each dimension. We modelled the optimization algorithm using an recurrent neural net with a single layer of 128 LSTM [71] cells. We used a time horizon of 400 iterations and a mini-batch size of 64 for computing stochastic gradients and objective values. We evaluate the optimization algorithm on its ability to generalize to unseen objective functions, which correspond to the problems of training neural nets on different tasks/datasets. We evaluate the learned optimization algorithm on three datasets, the Toronto Faces Dataset (TFD), CIFAR-10 and CIFAR-100. These datasets are chosen for their very different characteristics from MNIST and each other: TFD contains 3300 grayscale images that have relatively little variation and has seven different categories, whereas CIFAR-100 contains 50,000 colour images that have varied appearance and has 100 different categories.

All algorithms are tuned on the training objective function. For hand-engineered algorithms, this entails choosing the best hyperparameters; for learned algorithms, this entails meta-training on the objective function. We compare to the seven hand-engineered algorithms: stochastic gradient descent, momentum, conjugate gradient, L-BFGS, ADAM, AdaGrad and RMSprop. In addition, we compare to an optimization algorithm meta-trained using the method described in [4] on the same training objective function (training two-layer neural net on randomly projected and normalized MNIST) under the same setting (a time horizon of 400 iterations and a mini-batch size of 64).

First, we examine the performance of various optimization algorithms on similar objective functions. The optimization problems under consideration are those for training neural nets that have the same number of input and hidden units (48 and 48) as those used during meta-training. The number of output units varies with the number of categories in each dataset. We use the same mini-batch size as that used during meta-training. As shown



(a) (b) (c)

Figure 2.11: Comparison of the various hand-engineered and learned algorithms on training neural nets with 100 input units and 200 hidden units on (a) TFD, (b) CIFAR-10 and (c) CIFAR-100 for 800 iterations with mini-batches of size 64. The vertical axis is the true objective value and the horizontal axis represents the iteration. Best viewed in colour.

in Figure 2.7, the optimization algorithm meta-trained using our method (which we will refer to as Predicted Step Descent) consistently descends to the optimum the fastest across all datasets. On the other hand, other algorithms are not as consistent and the relative ranking of other algorithms varies by dataset. This suggests that Predicted Step Descent has learned to be robust to variations in the data distributions, despite being trained on only one objective function, which is associated with a very specific data distribution that characterizes MNIST. It is also interesting to note that while the algorithm meta-trained using [4] (which we will refer to as L2LBGDBGD) performs well on CIFAR, it is unable to reach the optimum on TFD.

Next, we change the architecture of the neural nets and see if Predicted Step Descent generalizes to the new architecture. We increase the number of input units to 100 and the number of hidden units to 200, so that the number of parameters is roughly increased by a factor of 8. As shown in Figure 2.8, Predicted Step Descent consistently outperforms other algorithms on each dataset, despite having not been trained to optimize neural nets of this architecture. Interestingly, while it exhibited a bit of oscillation initially on TFD and CIFAR-10, it quickly recovered and overtook other algorithms, which is reminiscent of the phenomenon for low-dimensional optimization problems. This suggests that it has learned to detect when it is performing poorly and knows how to change tack accordingly. L2LBGDBGD experienced difficulties on TFD and CIFAR-10 as well, but slowly diverged.

We now investigate how robust Predicted Step Descent is to stochasticity of the gradients. To this end, we take a look at its performance when we reduce the mini-batch size from 64 to 10 on both the original architecture with 48 input and hidden units and the enlarged architecture with 100 input units and 200 hidden units. As shown in Figure 2.9, on the original architecture, Predicted Step Descent still outperforms all other algorithms and is able to handle the increased stochasticity fairly well. In contrast, conjugate gradient and L2LBGDBGD had some difficulty handling the increased stochasticity on TFD and to a lesser extent, on CIFAR-10. In the former case, both diverged; in the latter case, both were

progressing slowly towards the optimum.

On the enlarged architecture, Predicted Step Descent experienced some significant oscillations on TFD and CIFAR-10, but still managed to achieve a much better objective value than all the other algorithms. Many hand-engineered algorithms also experienced much greater oscillations than previously, suggesting that the optimization problems are inherently harder. L2LBGDBGD diverged fairly quickly on these two datasets.

Finally, we try doubling the number of iterations. As shown in Figure 2.11, despite being trained over a time horizon of 400 iterations, Predicted Step Descent behaves reasonably beyond the number of iterations it is trained for.

Chapter 3

Implicit Maximum Likelihood Estimation

Generative modelling is a cornerstone of machine learning and has received increasing attention. Recent models like variational autoencoders (VAEs) [81, 109] and generative adversarial nets (GANs) [57, 62], have delivered impressive advances in performance and generated a lot of excitement.

Generative models can be classified into two categories: *prescribed models* and *implicit models* [43, 100]. Prescribed models are defined by an explicit specification of the density, and so their unnormalized complete likelihood can be usually expressed in closed form. Examples include models whose complete likelihoods lie in the exponential family, such as mixture of Gaussians [51], hidden Markov models [12], Boltzmann machines [69]. Because computing the normalization constant, also known as the partition function, is generally intractable, sampling from these models is challenging.

On the other hand, implicit models are defined most naturally in terms of a (simple) sampling procedure. Most models take the form of a deterministic parameterized transformation $T_\theta(\cdot)$ of an analytic distribution, like an isotropic Gaussian. This can be naturally viewed as the distribution induced by the following sampling procedure:

1. Sample $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$
2. Return $\mathbf{x} \coloneqq T_\theta(\mathbf{z})$

The transformation $T_\theta(\cdot)$ often takes the form of a highly expressive function approximator, like a neural net. Examples include generative adversarial nets (GANs) [57, 62] and generative moment matching nets (GMMNs) [92, 48]. The marginal likelihood of such models can be characterized as follows:

$$p_\theta(\mathbf{x}) = \frac{\partial}{\partial x_1} \cdots \frac{\partial}{\partial x_d} \int_{\left\{\mathbf{z} \middle| \forall i \ (T_\theta(\mathbf{z}))_i \leq x_i\right\}} \phi(\mathbf{z}) d\mathbf{z}$$

where $\phi(\cdot)$ denotes the probability density function (PDF) of $\mathcal{N}(0, \mathbf{I})$.

In general, attempting to reduce this to a closed-form expression is hopeless. Evaluating it numerically is also challenging, since the domain of integration could consist of an exponential number of disjoint regions and numerical differentiation is ill-conditioned.

These two categories of generative models are not mutually exclusive. Some models admit both an explicit specification of the density and a simple sampling procedure and so can be considered as both prescribed and implicit. Examples include variational autoencoders [81, 109], their predecessors [96, 29] and extensions [33], and directed/autoregressive models, e.g., [102, 18, 84, 104].

3.1 Challenges in Parameter Estimation

Maximum likelihood [54, 49] is perhaps the standard method for estimating the parameters of a probabilistic model from observations. The maximum likelihood estimator (MLE) has a number of appealing properties: under mild regularity conditions, it is asymptotically consistent, efficient and normal. A long-standing challenge of training probabilistic models is the computational roadblocks of maximizing the log-likelihood function directly.

For prescribed models, maximizing likelihood directly requires computing the partition function, which is intractable for all but the simplest models. Many powerful techniques have been developed to attack this problem, including variational methods [79], contrastive divergence [68, 126], score matching [75] and pseudolikelihood maximization [26], among others.

For implicit models, the situation is even worse, as there is no term in the log-likelihood function that is in closed form; evaluating any term requires computing an intractable integral. As a result, maximizing likelihood in this setting seems hopelessly difficult. A variety of likelihood-free solutions have been proposed that in effect minimize a divergence measure between the data distribution and the model distribution. They come in two forms: those that minimize an f-divergence, and those that minimize an integral probability metric [101]. In the former category are GANs, which are based on the idea of minimizing the distinguishability between data and samples [121, 63]. It has been shown that when given access to an infinitely powerful discriminator, the original GAN objective minimizes the Jensen-Shannon divergence, the $-\log D$ variant of the objective minimizes the reverse KL-divergence minus a bounded quantity [5], and later extensions [103] minimize arbitrary f-divergences. In the latter category are GMMNs which use maximum mean discrepancy (MMD) [60] as the witness function.

In the case of GANs, despite the theoretical results, there are a number of challenges that arise in practice, such as mode dropping/collapse [57, 7], vanishing gradients [5, 116] and training instability [57, 8]. A number of explanations have been proposed to explain these phenomena and point out that many theoretical results rely on three assumptions: the discriminator must have infinite modelling capacity [57, 8], the number of samples from the true data distribution must be infinite [8, 116] and the gradient ascent-descent procedure [9, 113] can converge to a global pure-strategy Nash equilibrium [57, 8]. When some of these

assumptions do not hold, the theoretical guarantees do not necessarily apply. A number of ways have been proposed that alleviate some of these issues, e.g., [129, 112, 45, 47, 6, 70, 93, 132], but a way of solving all three issues simultaneously remains elusive.

3.2 Contribution

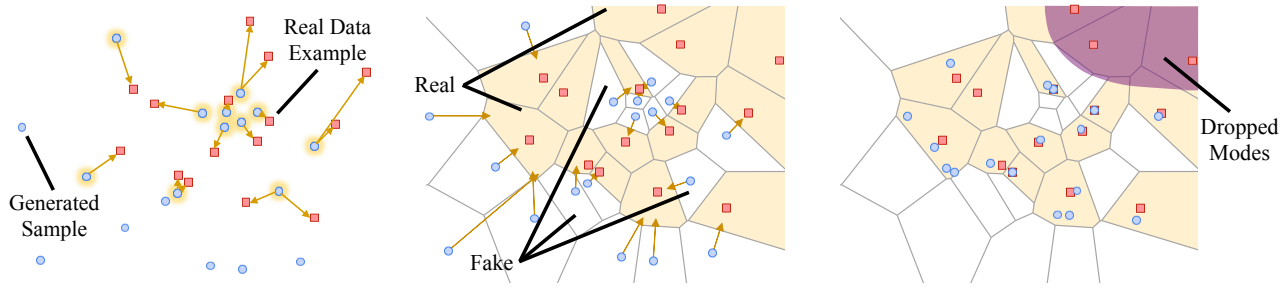
We present an alternative method for estimating parameters in implicit models. Like the methods above, our method is likelihood-free, but can be shown to be equivalent to maximizing likelihood under some conditions. Our result holds in the parametric finite sample setting, i.e.: when the capacity of the model is finite and the number of data examples is finite. The idea behind the method is simple: it finds the nearest *generated sample* to each *data example* and optimizes the model parameters to pull the sample towards it. The direction in which nearest neighbour search is performed is of *critical importance* – we contrast this with a GAN with a 1-nearest neighbour discriminator in Figure 3.1, and show that such a model essentially performs nearest neighbour search in the other direction, i.e.: push each *generated sample* to the nearest *data example*. The latter ensures that each sample is close to a data example, but each data example does not necessarily have a nearby sample. Intuitively, the data examples that do not have nearby samples are not assigned high density by the model – in other words, the modes that generated the data examples are dropped.

The proposed method could sidestep the three issues mentioned above: mode collapse, vanishing gradients and training instability. Modes are not dropped because the loss ensures each data example has a sample nearby at optimality; gradients do not vanish because the gradient of the distance between a data example and its nearest sample does not become zero unless they coincide; training is stable because the estimator is the solution to a simple minimization problem. By leveraging recent advances in fast nearest neighbour search algorithms [90, 91], this approach is able to scale to large, high-dimensional datasets.

3.3 Method

3.3.1 Intuition

Consider a model distribution that maximizes the likelihood of the data. Since likelihood is the product of densities evaluated at all data examples, the model density at each data example should be high. Suppose we don't observe the model distribution directly, and instead only observe independent and identically distributed (i.i.d.) samples drawn from the model. Because the density at data examples is high, more samples are expected to lie near data examples than elsewhere. Can we construct an objective function that encourages the model to have this behaviour? A lot of samples near a data example typically means that radius of the neighbourhood around the data example that only contains one sample is small. Therefore, a natural objective is to minimize the distance from each data example to the nearest sample.



(a) IMLE (Proposed Method) (b) Nearest Neighbours GAN (Step 1) (c) Nearest Neighbours GAN (Step 2)

Figure 3.1: (a) An illustration of how the proposed method works, and (b-c) a comparison to a GAN with a 1-nearest neighbour discriminator. The blue circles represent generated samples and the red squares represent real data examples. In (b-c), the yellow regions represent those classified as real by the discriminator, whereas the white regions represent those classified as fake. In the case of (a) the proposed method (IMLE), each *data example* pulls the nearest *sample* towards it, whereas in the case of (b-c) the GAN, each *sample* is essentially pushed towards the nearest *data example*. In the latter case, some data examples may not be selected by any sample and therefore will not have samples nearby – this is a manifestation of mode dropping, since the modes that generated these data examples are not modelled. The proposed method avoids this phenomenon because it conducts nearest neighbour search in the opposite direction, which ensures that every data example will have a nearby sample.

3.3.2 Definition

We are given a set of n data examples $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and some unknown parameterized probability distribution P_θ with density p_θ. We also have access to an oracle that allows us to draw independent and identically distributed (i.i.d.) samples from P_θ.

Let $\tilde{\mathbf{x}}_1^\theta, \ldots, \tilde{\mathbf{x}}_m^\theta$ be i.i.d. samples from P_θ, where $m \geq n$. For each data example $\mathbf{x}_i$, we define a random variable R_i^θ to be the distance between $\mathbf{x}_i$ and the nearest sample. More precisely,

$$R_i^\theta \coloneqq \min_{j \in [m]} \left\| \tilde{\mathbf{x}}_j^\theta - \mathbf{x}_i \right\|_2^2$$

where $[m]$ denotes $\{1, \ldots, m\}$.

The implicit maximum likelihood estimator $\hat{\theta}_{\text{IMLE}}$ is defined as:

$$\begin{aligned} \hat{\theta}_{\text{IMLE}} &\coloneqq \arg\min_\theta \mathbb{E}_{R_1^\theta, \ldots, R_n^\theta} \left[\sum_{i=1}^n R_i^\theta \right] \\ &= \arg\min_\theta \mathbb{E}_{\tilde{\mathbf{x}}_1^\theta, \ldots, \tilde{\mathbf{x}}_m^\theta} \left[\sum_{i=1}^n \min_{j \in [m]} \left\| \tilde{\mathbf{x}}_j^\theta - \mathbf{x}_i \right\|_2^2 \right] \end{aligned}$$

An illustration of how the proposed method works is shown in Figure 3.1, along with a comparison to a GAN with a 1-nearest neighbour discriminator.

3.3.3 Algorithm

We outline the proposed parameter estimation procedure in Algorithm 6. In each outer iteration, we draw m i.i.d. samples from the current model P_θ. We then randomly select a batch of examples from the dataset and find the nearest sample from each data example. We then run a standard iterative optimization algorithm, like stochastic gradient descent (SGD), to minimize a sample-based version of the Implicit Maximum Likelihood Estimator (IMLE) objective.

Because our algorithm needs to solve a nearest neighbour search problem in each outer iteration, the scalability of our method depends on our ability to find the nearest neighbours quickly. This was traditionally considered to be a hard problem, especially in high dimensions. However, this is no longer the case, due to recent advances in nearest neighbour search algorithms [90, 91].

Algorithm 6 Implicit maximum likelihood estimation (IMLE) procedure

Require: The dataset $D = \{\mathbf{x}_i\}_{i=1}^n$ and a sampling mechanism for the implicit model P_θ
  Initialize θ to a random vector
  for $k = 1$ **to** K **do**
    Draw i.i.d. samples $\tilde{\mathbf{x}}_1^\theta, \ldots, \tilde{\mathbf{x}}_m^\theta$ from P_θ
    Pick a random batch $S \subseteq \{1, \ldots, n\}$
    $\sigma(i) \leftarrow \arg\min_j \left\|\mathbf{x}_i - \tilde{\mathbf{x}}_j^\theta\right\|_2^2 \ \forall i \in S$
    for $l = 1$ **to** L **do**
      Pick a random mini-batch $\tilde{S} \subseteq S$
      $\theta \leftarrow \theta - \eta \nabla_\theta \left(\frac{n}{|\tilde{S}|} \sum_{i \in \tilde{S}} \left\|\mathbf{x}_i - \tilde{\mathbf{x}}_{\sigma(i)}^\theta\right\|_2^2 \right)$
    end for
  end for
  return θ

Note that the use of Euclidean distance is not a major limitation of the proposed approach. A variety of distance metrics are either exactly or approximately equivalent to Euclidean distance in some non-linear embedding space, in which case the theoretical guarantees are inherited from the Euclidean case. This encompasses popular distance metrics used in the literature, like the Euclidean distance between the activations of a neural net, which is often referred to as a perceptual similarity metric [112, 46]. For distance metrics that cannot be embedded in Euclidean space, the analysis can be easily adapted to other metrics with minor modifications.

3.4 Analysis

Before formally stating the theoretical results, we first illustrate the intuition behind why the proposed estimator is equivalent to maximum likelihood estimator under some conditions. For simplicity, we will consider the special case where we only have a single data example $\mathbf{x}_1$ and a single sample $\tilde{\mathbf{x}}_1^\theta$. Consider the total density of P_θ inside a ball of radius of t centred at $\mathbf{x}_1$ as a function of t, a function that will be denoted as $\tilde{F}^\theta(t)$. If the density in the neighbourhood of $\mathbf{x}_1$ is high, then $\tilde{F}^\theta(t)$ would grow rapidly as t increases. If, on the other hand, the density in the neighbourhood of $\mathbf{x}_1$ is low, then $\tilde{F}^\theta(t)$ would grow slowly. So, maximizing likelihood is equivalent to making $\tilde{F}^\theta(t)$ grow as fast as possible. To this end, we can maximize the area under the function $\tilde{F}^\theta(t)$, or equivalently, minimize the area under the function $1 - \tilde{F}^\theta(t)$. Observe that $\tilde{F}^\theta(t)$ can be interpreted as the cumulative distribution function (CDF) of the Euclidean distance between $\mathbf{x}_1$ and $\tilde{\mathbf{x}}_1^\theta$, which is a random variable because $\tilde{\mathbf{x}}_1^\theta$ is random and will be denoted as $\tilde{R}^\theta$. Because $\tilde{R}^\theta$ is non-negative, recall that $\mathbb{E}\left[\tilde{R}^\theta\right] = \int_0^\infty \Pr\left(\tilde{R}^\theta > t\right) dt = \int_0^\infty \left(1 - \tilde{F}^\theta(t)\right) dt$, which is exactly the area under the function $1 - \tilde{F}^\theta(t)$. Therefore, we can maximize likelihood of a data example $\mathbf{x}_1$ by minimizing $\mathbb{E}\left[\tilde{R}^\theta\right]$, or in other words, minimizing the expected distance between the data example and a random sample. To extend this analysis to the case with multiple data examples, we show in the appendix that if the objective function is a summation, applying a monotonic transformation to each term and then reweighting appropriately preserves the optimizer under some conditions.

We now state the key theoretical result formally. Please refer to the appendix for the proof.

Theorem 15. *Consider a set of observations $\mathbf{x}_1, \ldots, \mathbf{x}_n$, a parameterized family of distributions P_θ with probability density function (PDF) $p_\theta(\cdot)$ and a unique maximum likelihood solution θ^*. For any $m \geq 1$, let $\tilde{\mathbf{x}}_1^\theta, \ldots, \tilde{\mathbf{x}}_m^\theta \sim P_\theta$ be i.i.d. random variables and define $\tilde{r}^\theta \coloneqq \left\|\tilde{\mathbf{x}}_1^\theta\right\|_2^2$, $R^\theta \coloneqq \min_{j\in[m]} \left\|\tilde{\mathbf{x}}_j^\theta\right\|_2^2$ and $R_i^\theta \coloneqq \min_{j\in[m]} \left\|\tilde{\mathbf{x}}_j^\theta - \mathbf{x}_i\right\|_2^2$. Let $F^\theta(\cdot)$ be the cumulative distribution function (CDF) of $\tilde{r}^\theta$ and $\Psi(z) \coloneqq \min_\theta \left\{\mathbb{E}\left[R^\theta\right] | p_\theta(\mathbf{0}) = z\right\}$.*

If P_θ satisfies the following:

- *$p_\theta(\mathbf{x})$ is differentiable w.r.t. θ and continuous w.r.t. $\mathbf{x}$ everywhere.*
- *$\forall \theta, \mathbf{v}$, there exists θ' such that $p_\theta(\mathbf{x}) = p_{\theta'}(\mathbf{x} + \mathbf{v})\ \forall \mathbf{x}$.*
- *For any θ_1, θ_2, there exists θ_0 such that $F^{\theta_0}(t) \geq \max\left\{F^{\theta_1}(t), F^{\theta_2}(t)\right\}\ \forall t \geq 0$ and $p_{\theta_0}(\mathbf{0}) = \max\left\{p_{\theta_1}(\mathbf{0}), p_{\theta_2}(\mathbf{0})\right\}$.*
- *$\exists \tau > 0$ such that $\forall i \in [n]\ \forall \theta \notin B_{\theta^*}(\tau)$, $p_\theta(\mathbf{x}_i) < p_{\theta^*}(\mathbf{x}_i)$, where $B_{\theta^*}(\tau)$ denotes the ball centred at θ^* of radius τ.*
- *$\Psi(z)$ is differentiable everywhere.*

- *For all θ, if $\theta \neq \theta^*$, there exists $j \in [d]$ such that*
$$\left\langle \begin{pmatrix} \frac{\Psi'(p_\theta(\mathbf{x}_1))p_\theta(\mathbf{x}_1)}{\Psi'(p_{\theta^*}(\mathbf{x}_1))p_{\theta^*}(\mathbf{x}_1)} \\ \vdots \\ \frac{\Psi'(p_\theta(\mathbf{x}_n))p_\theta(\mathbf{x}_n)}{\Psi'(p_{\theta^*}(\mathbf{x}_n))p_{\theta^*}(\mathbf{x}_n)} \end{pmatrix}, \begin{pmatrix} \nabla_\theta \left(\log p_\theta(\mathbf{x}_1)\right)_j \\ \vdots \\ \nabla_\theta \left(\log p_\theta(\mathbf{x}_n)\right)_j \end{pmatrix} \right\rangle \neq 0.$$

Then,

$$\arg\min_\theta \sum_{i=1}^n \frac{\mathbb{E}\left[R_i^\theta\right]}{\Psi'(p_{\theta^*}(\mathbf{x}_i))p_{\theta^*}(\mathbf{x}_i)} = \arg\max_\theta \sum_{i=1}^n \log p_\theta(\mathbf{x}_i)$$

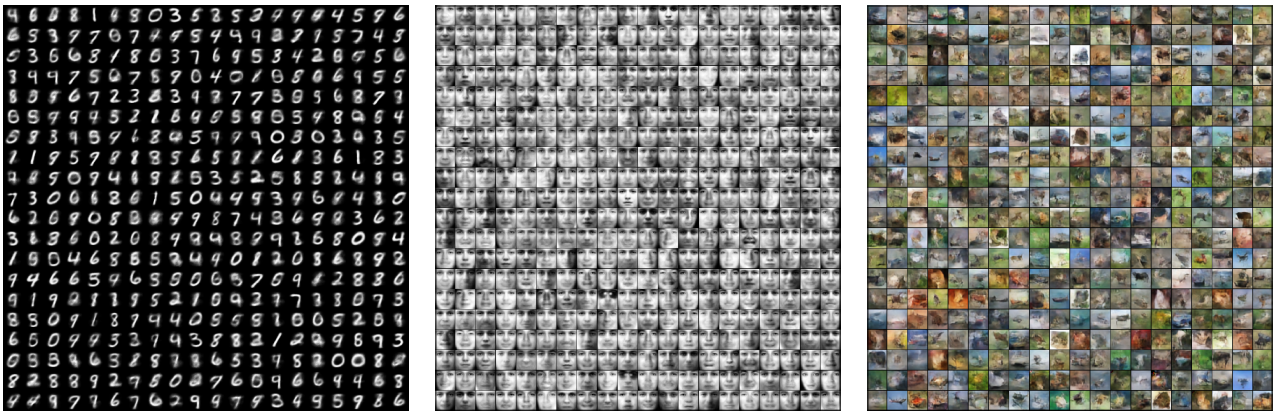
Furthermore, if $p_{\theta^*}(\mathbf{x}_1) = \cdots = p_{\theta^*}(\mathbf{x}_n)$, then,

$$\arg\min_\theta \sum_{i=1}^n \mathbb{E}\left[R_i^\theta\right] = \arg\max_\theta \sum_{i=1}^n \log p_\theta(\mathbf{x}_i)$$

Now, we examine the restrictiveness of each condition. The first condition is satisfied by nearly all analytic distributions. The second condition is satisfied by nearly all distributions that have an unrestricted location parameter, since one can simply shift the location parameter by $\mathbf{v}$. The third condition is satisfied by most distributions that have location and scale parameters, like a Gaussian distribution, since the scale can be made arbitrarily low and the location can be shifted so that the constraint on $p_\theta(\cdot)$ is satisfied. The fourth condition is satisfied by nearly all distributions, whose density eventually tends to zero as the distance from the optimal parameter setting tends to infinity. The fifth condition requires $\min_\theta \left\{\mathbb{E}\left[R^\theta\right] | p_\theta(\mathbf{0}) = z\right\}$ to change smoothly as $\mathbf{z}$ changes. The final condition requires the two n-dimensional vectors, one of which can be chosen from a set of d vectors, to be not exactly orthogonal. As a result, this condition is usually satisfied when d is large, i.e. when the model is richly parameterized.

There is one remaining difficulty in applying this theorem, which is that the quantity $1/\Psi'(p_{\theta^*}(\mathbf{x}_i))p_{\theta^*}(\mathbf{x}_i)$, which appears as an coefficient on each term in the proposed objective, is typically not known. If we consider a new objective that ignores the coefficients, i.e. $\sum_{i=1}^n \mathbb{E}\left[R_i^\theta\right]$, then minimizing this objective is equivalent to minimizing an upper bound on the ideal objective, $\sum_{i=1}^n \mathbb{E}\left[R_i^\theta\right] / \Psi'(p_{\theta^*}(\mathbf{x}_i))p_{\theta^*}(\mathbf{x}_i)$. The tightness of this bound depends on the difference between the highest and lowest likelihood assigned to individual data points at the optimum, i.e. the maximum likelihood estimate of the parameters. Such a model should not assign high likelihoods to some points and low likelihoods to others as long as it has reasonable capacity, since doing so would make the overall likelihood, which is the product of the likelihoods of individual data points, low. Therefore, the upper bound is usually reasonably tight.

3.5 Experiments



(a) MNIST (b) TFD (c) CIFAR-10

Figure 3.2: Representative random samples from the model trained on (a) MNIST, (b) Toronto Faces Dataset and (c) CIFAR-10.

We trained generative models using the proposed method on three standard benchmark datasets, MNIST, the Toronto Faces Dataset (TFD) and CIFAR-10. All models take the form of feedforward neural nets with isotropic Gaussian noise as input.

Method	MNIST	TFD
DBN [19]	138 ± 2	1909 ± 66
SCAE [19]	121 ± 1.6	2110 ± 50
DGSN [20]	214 ± 1.1	1890 ± 29
GAN [57]	225 ± 2	2057 ± 26
GMMN [92]	147 ± 2	2085 ± 25
IMLE (Proposed)	$\mathbf{257 \pm 6}$	$\mathbf{2139 \pm 27}$

Table 3.1: Log-likelihood of the test data under the Gaussian Parzen window density estimated from samples generated by different methods.

For MNIST, the architecture consists of two fully connected hidden layers with 1200 units each followed by a fully connected output layer with 784 units. ReLU activations were used for hidden layers and sigmoids were used for the output layer. For TFD, the architecture is wider and consists of two fully connected hidden layers with 8000 units each followed by a fully connected output layer with 2304 units. For both MNIST and TFD, the dimensionality of the noise vector is 100.

For CIFAR-10, we used a simple convolutional architecture with 1000-dimensional Gaussian noise as input. The architecture consists of five convolutional layers with 512 output channels and a kernel size of 5 that all produce 4×4 feature maps, followed by a bilinear upsampling layer that doubles the width and height of the feature maps. There is a batch normalization layer followed by leaky ReLU activations with slope -0.2 after each convolutional layer. This design is then repeated for each subsequent level of resolution, namely 8×8, 16×16 and 32×32, so that we have 20 convolutional layers, each with output 512 channels. We then add a final output layer with three output channels on top, followed by sigmoid activations. We note that this architecture has more capacity than typical archi-

tectures used in other methods, like [108]. This is because our method aims to capture all modes of the data distribution and therefore needs more modelling capacity than methods that are permitted to drop modes.

Evaluation for implicit generative models in general remains an open problem. Extrinsic evaluation metrics measure performance indirectly via a downstream task, e.g.: Inception score or Fréchet Inception distance [112, 67]. Unfortunately, dependence on the downstream task could introduce bias.

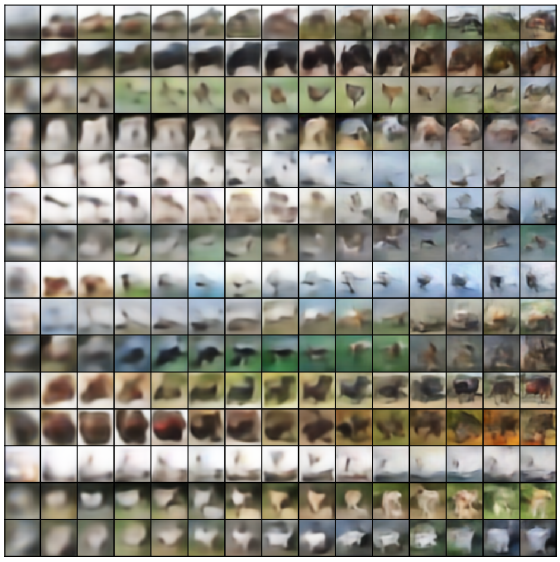


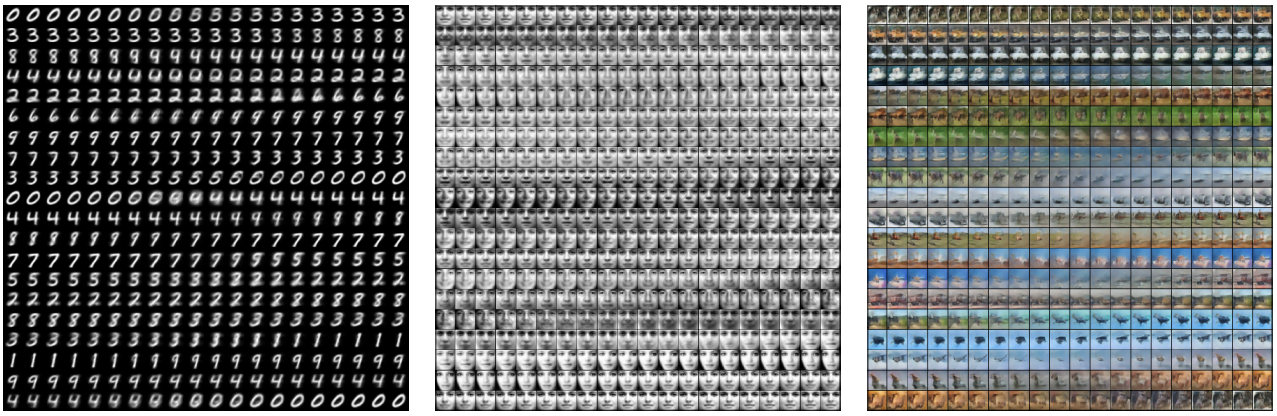
Figure 3.3: Samples corresponding to the same latent variable values at different points in time while training the model on CIFAR-10. Each row corresponds to a sample, and each column corresponds to a particular point in time.

Intrinsic evaluation metrics measure performance without relying on external models or data. Popular examples like estimated log-likelihood [20, 127] and visual assessment of sample quality evaluate different properties – sample quality reflects precision, whereas estimated log-likelihood focuses on recall. Consequently, one is not a replacement for the other. Two models that achieve different levels of precision may simply be at different points on the same precision-recall curve, and therefore may not be directly comparable. We visualize randomly chosen samples in Figure 3.2 and report the estimated log-likelihood in Table 3.1. As mentioned above, both evaluation criteria have biases/deficiencies, so performing well on either of these metrics does not necessarily indicate good density estimation performance. However, not performing badly on either metric can provide some comfort that the model is simultaneously able to achieve reasonable precision and recall.

As shown in Figure 3.2, despite its simplicity, the proposed method is able to generate reasonably good samples for MNIST, TFD and CIFAR-10. Samples also seem fairly diverse. This is supported by the estimated log-likelihood results in Table 3.1. Because the model achieved a high score on that metric on both MNIST and TFD, this suggests that the model did not suffer from significant mode dropping.

In Figure 3.5, we show samples and their nearest neighbours in the training set. Each sample is quite different from its nearest neighbour in the training set, suggesting that the model has not overfitted to examples in the training set.

Next, we visualize the learned manifold by walking along a geodesic on the manifold between pairs of samples. More concretely, we generate five samples, arrange them in arbitrary order, perform linear interpolation in latent variable space between adjacent pairs



(a) MNIST (b) TFD (c) CIFAR-10

Figure 3.4: Linear interpolation between samples in latent code space. The first image in every row is an independent sample; all other images are interpolated between the previous and the subsequent sample. Images along the path of interpolation are shown in the figure arranged from left to right, top to bottom. They also wrap around, so that images in the last row are interpolations between the last and first samples.

of samples, and generate an image from the interpolated latent variable. As shown in Figure 3.14, the images along the path of interpolation appear visually plausible and do not have noisy artifacts. In addition, the transition from one image to the next appears smooth on all datasets. This indicates that the support of the model distribution has not collapsed to a set of isolated points and that the proposed method is able to learn the geometry of the data manifold.

Finally, we illustrate the evolution of samples as training progresses in Figure 3.3. As shown, the samples are initially blurry and become sharper over time. Importantly, sample quality consistently improves over time, which demonstrates the stability of training.

3.6 Conditional Generative Modelling

In conditional generative modelling, the goal is to model the conditional distribution $p(\mathbf{y}|\mathbf{x})$, rather than the marginal distribution $p(\mathbf{y})$.

We consider an implicit model that is defined by the following sampling procedure:

1. Sample $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$
2. Return $\mathbf{y} := T_\theta(\mathbf{x}, \mathbf{z})$

Here, T_θ is a neural net that takes in two inputs, the input $\mathbf{x}$ and the latent noise vector $\mathbf{z}$. We can contrast this with the unconditional setting, where T_θ only takes in a single input, the latent noise vector $\mathbf{z}$.

(a) MNIST (b) TFD (c) CIFAR-10

Figure 3.5: Comparison of samples and their nearest neighbours in the training set. Images in odd-numbered columns are samples; to the right of each sample is its nearest neighbour in the training set.

In the conditional setting, we are effectively trying to model a family of distributions, each of which is associated with a different value of $\mathbf{x}$. As a result, two changes need to be made to the IMLE algorithm. First, the value of $\mathbf{x}$ must be provided to the transformation function T_θ in order to sample from the correct distribution. Second, the samples for different values of $\mathbf{x}$ must be kept separate since they are from different distributions. Consequently, for each input data example $\mathbf{x}_i$, we must look for the nearest sample in among the samples generated from $p(\tilde{\mathbf{y}}|\mathbf{x}_i)$. The IMLE algorithm modified for the conditional setting, which will be known as conditional IMLE, is presented in Algorithm 7.

We also generalize the distance metric that is used, and modify the loss so that it admits an arbitrary metric $\mathcal{L}(\cdot, \cdot)$ between ground truth data and generated samples.

3.7 Application to Multimodal Conditional Image Synthesis

In conditional image synthesis, the goal is to generate an image from some input, which can influence the image that is generated. It encompasses a broad range of tasks; examples include super-resolution, which aims to generate high-resolution images from low-resolution inputs, and image synthesis from scene layout, which aims to generate images from semantic segmentation maps.

Predominant approaches focus on the setting of generating a single image for each input image, which we will refer to as the *unimodal prediction* problem. Relatively less attention has been devoted to the more general and challenging problem of *multimodal prediction*, which aims to generate multiple equally plausible images for the *same* input image.

Algorithm 7 Conditional Implicit Maximum Likelihood Estimation (IMLE) Procedure

Require: The set of input $X = \{\mathbf{x}_i\}_{i=1}^n$, the set of corresponding targets $Y = \{\mathbf{y}_i\}_{i=1}^n$ and a distance metric $\mathcal{L}(\cdot, \cdot)$
  Initialize the parameters θ of the model/transformation function T_θ
  for $p = 1$ **to** N **do**
    Pick a random batch $S \subseteq \{1, \ldots, n\}$
    for $i \in S$ **do**
      Randomly generate i.i.d. m noise vectors $\mathbf{z}_1, \ldots, \mathbf{z}_m$
      $\tilde{\mathbf{y}}_{i,j} \leftarrow T_\theta(\mathbf{x}_i, \mathbf{z}_j)\ \forall j \in [m]$
      $\sigma(i) \leftarrow \arg\min_j \mathcal{L}(\mathbf{y}_i, \tilde{\mathbf{y}}_{i,j})\ \forall j \in [m]$
    end for
    for $q = 1$ **to** M **do**
      Pick a random mini-batch $\tilde{S} \subseteq S$
      $\theta \leftarrow \theta - \eta \nabla_\theta \left(\sum_{i \in \tilde{S}} \mathcal{L}(\mathbf{y}_i, \tilde{\mathbf{y}}_{i,\sigma(i)}) \right) / |\widetilde{S}|$
    end for
  end for
  return θ

Why is the latter important? Conditional image synthesis is, by its very nature, ill-posed. That is, the information in the input is not enough to fully constrain the degrees of freedom in the output, and there are many plausible outputs that could all be consistent with the input. Therefore, we would like our system to be capable of generating all plausible outputs, rather than just selecting some plausible output arbitrarily. This could be important for downstream applications; for example, we may need to know the uncertainty of our system to estimate the informativeness of the input, or filter out a subset of the generated images to conform to some user-specified constraint.

This can be naturally formulated as a conditional generative modelling problem, where $\mathbf{x}$ corresponds to the input image and $\mathbf{y}$ corresponds to the output image. The distribution we would like to model is therefore $p(\mathbf{y}|\mathbf{x})$. Each plausible output image that is consistent with the input image would be a mode of the distribution; because there could be many plausible images that are consistent with the same input image, $p(\mathbf{y}|\mathbf{x})$ is usually multimodal. Different output images could be then generated by sampling from our model for this distribution.

We demonstrate that our method is able to generate arbitrarily many different images for the same input image that are both diverse and faithful to the input image, which compares favourably conditional GAN-based methods, which can only generate a single output image for the same input image because of mode collapse.

3.7.1 Tasks

We consider two different conditional image synthesis tasks, single image super-resolution and image synthesis from scene layout. We describe them below.

(a) Input (b) Samples

Figure 3.6: Samples generated by the proposed method (known as super-resolution implicit model, or SRIM for short) for the task of single image super-resolution (by a factor of 8). The top row shows different samples generated by our method, and the bottom row shows the difference between adjacent samples. As shown by the difference between the samples, the proposed method is able to generate diverse samples.

3.7.2 Single Image Super-Resolution

Given a low-resolution image, the problem of super-resolution requires the prediction of multiple plausible versions of the image at a higher resolution. More formally, given a low-resolution image $\mathbf{x} \in [0, 255]^{h \times w \times 3}$, the goal is to predict plausible high-resolution images $\tilde{\mathbf{y}} \in [0, 255]^{H \times W \times 3}$ that when downsampled, are consistent with $\mathbf{x}$, where $H > h$ and $W > w$. We consider the challenging setting of upscaling by a factor of 8, i.e.: $H = 8h$ and $W = 8w$. In comparison, most prior super-resolution methods can only produce reasonable results when the upscaling factor is 4 or less.

3.7.3 Image Synthesis from Scene Layout

Given a semantic segmentation map, the goal is to generate multiple plausible images that are all consistent with the segmentation. More formally, given a segmentation map $L \in \{0, 1\}^{h \times w \times c}$ where $h \times w$ is the size of the image and c is the number of semantic classes, the goal is to generate plausible images $\widetilde{I} \in \mathbb{R}^{h \times w \times 3}$ that are consistent with L.

Some examples of the results are shown in Figures 3.6 and 3.7.

3.7.4 Comparison to Conditional GAN

Extending GAN-based approaches to perform multimodal prediction has proven to be challenging, due to the problem of *mode collapse*. More specifically, there is only one ground truth output for every input and the GAN objective encourages *every* generated sample based on an input to be similar to the corresponding ground truth output. As a result, the

(a) Input (b) Samples

Figure 3.7: Samples generated by the proposed method for the task of image synthesis from scene layout. The group of images on the right are the different samples generated by our method.

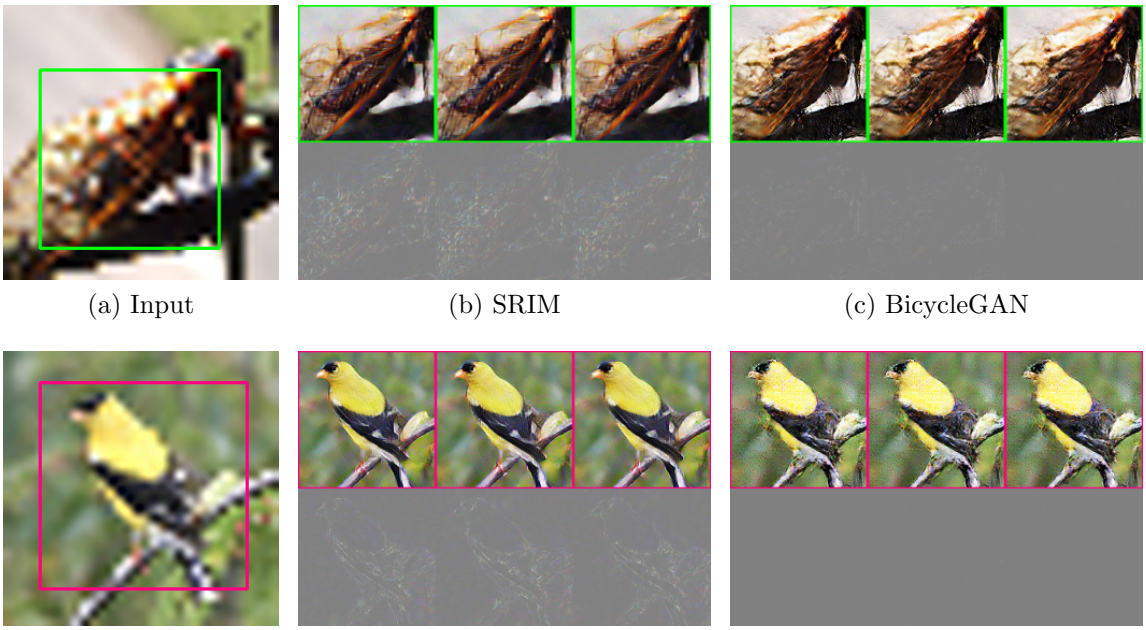
generator tends to produce almost identical samples for the same input, regardless of the noise vector that is fed in [77].

Intuitively, this problem occurs because *every* sample is made similar to the ground truth. This is undesirable, because there could be other images different from the ground truth that are also perfectly valid, due to the ill-posed nature of conditional image synthesis. Yet, generating any of such images would be penalized by the GAN objective; as a result, diversity is discouraged and mode collapse happens as a consequence. On the other hand, in conditional IMLE, only *one* of the samples is made similar to the ground truth.

3.7.5 Data

Single Image Super-Resolution For the problem of super-resolution, we can generate the training data by taking high-resolution images and downsampling them. The downsampled images will serve as input, and the original images will serve as the ground truth. The data is a subset of the ImageNet ILSVRC-2012 dataset. The ground truth images have a resolution of 256×256, which are obtained by anisotropic scaling of the original images. The input images have a resolution of 32×32, which are obtained by downsampling the 256×256 images. The images used for training and testing are disjoint.

Image Synthesis from Scene Layout The choice of dataset is important for multimodal image synthesis. For this task, the most common dataset in the unimodal setting is the Cityscapes dataset [37]. However, it is not suitable for the multimodal setting because most images in the dataset are taken under similar weather conditions and time of day and the amount of variation in object colours is limited. This lack of diversity limits what any multimodal method can do. On the other hand, the GTA-5 dataset [110], has much greater variation in terms of weather conditions and object appearance. To demonstrate this, we compare the colour distribution of both datasets and present the distribution of hues of both



(a) Input (b) SRIM (c) BicycleGAN

(d) Input (e) SRIM (f) BicycleGAN

Figure 3.8: Samples generated by the proposed method (SRIM) and the baseline (BicycleGAN). The top row in each group of images shows different samples generated by each method, and the bottom row shows the difference between adjacent samples. As shown in the bottom row, the difference between the samples of SRIM is greater than that of BicycleGAN, which indicates that SRIM is able to generate more diverse samples.

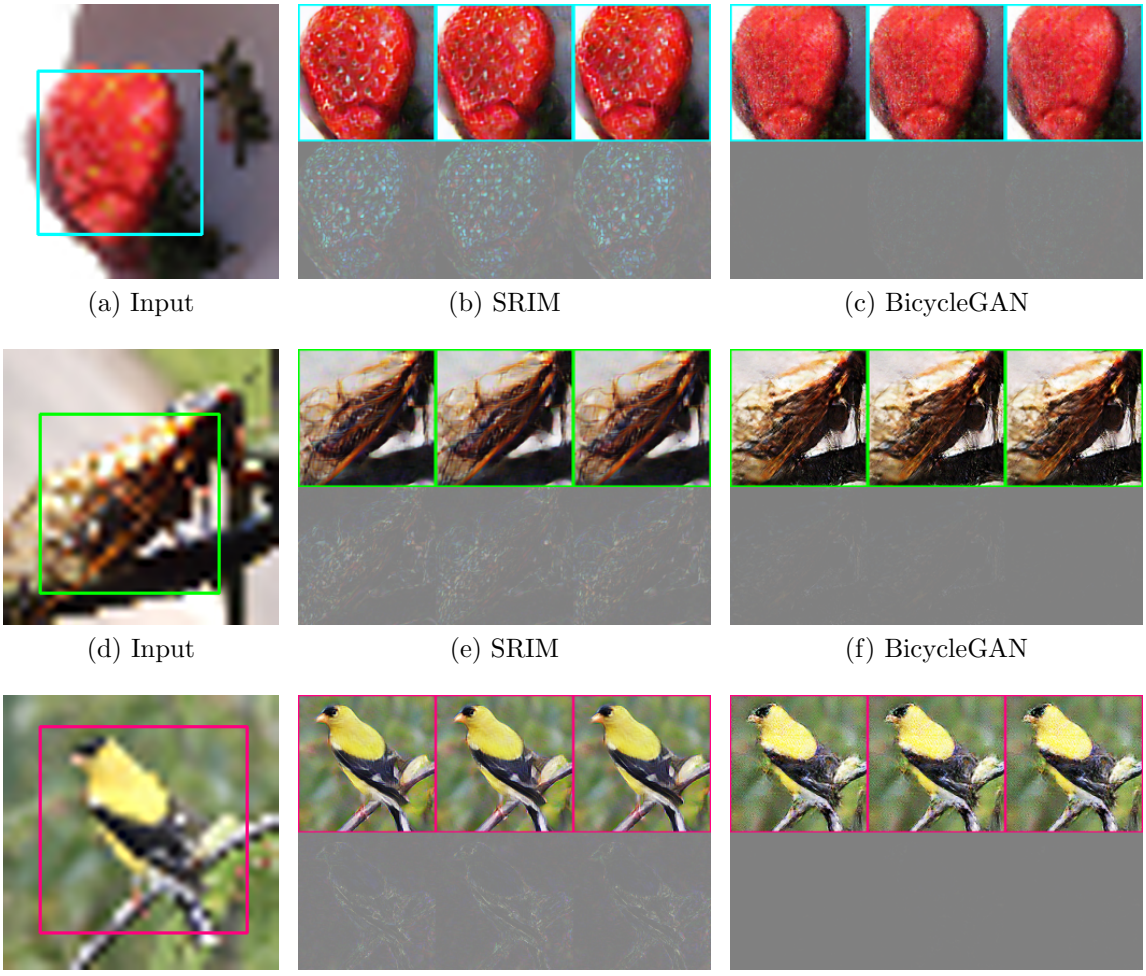
datasets in Figure 3.10. As shown, Cityscapes is concentrated around a single mode in terms of hue, whereas GTA-5 has much greater variation in hue. Additionally, the GTA-5 dataset includes more 20000 images and so is much larger than Cityscapes.

3.7.6 Implementation Details

Common Across Both Tasks For both tasks of super-resolution and image synthesis from semantic layout, we use a distance metric of the following form, where $\mathbf{y}$ and $\tilde{\mathbf{y}}$ denote the ground truth example and the generated sample respectively:

$$\mathcal{L}(\mathbf{y}, \tilde{\mathbf{y}}) = \sum_{i=1}^{l} \lambda_i ||\Phi_i(\mathbf{y}) - \Phi_i(\tilde{\mathbf{y}})||_p \tag{3.1}$$

Here, $\Phi_1(\cdot), \cdots, \Phi_l(\cdot)$ compute features of the function inputs. Hyperparameters $\{\lambda_i\}_{i=1}^{l}$ are set such that each term makes the same contribution to the total sum on average.



(a) Input (b) SRIM (c) BicycleGAN

(d) Input (e) SRIM (f) BicycleGAN

(g) Input (h) SRIM (i) BicycleGAN

Figure 3.9: Samples generated by the proposed method (SRIM) and the baseline (BicycleGAN). The top row in each group of images shows different samples generated by each method, and the bottom row shows the difference between adjacent samples. As shown in the bottom row, the difference between the samples of SRIM is greater than that of BicycleGAN, which indicates that SRIM is able to generate more diverse samples.

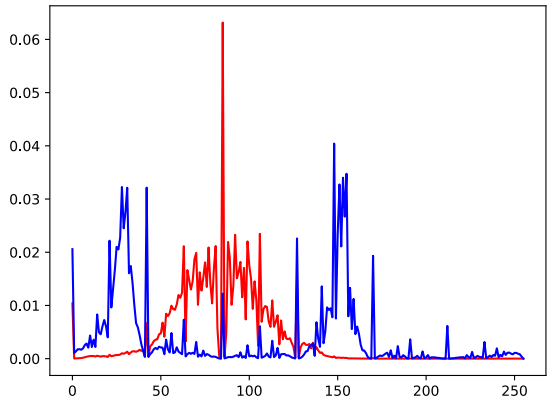


Figure 3.10: Comparison of histogram of hues between two datasets. Red is Cityscapes and blue is GTA-5.

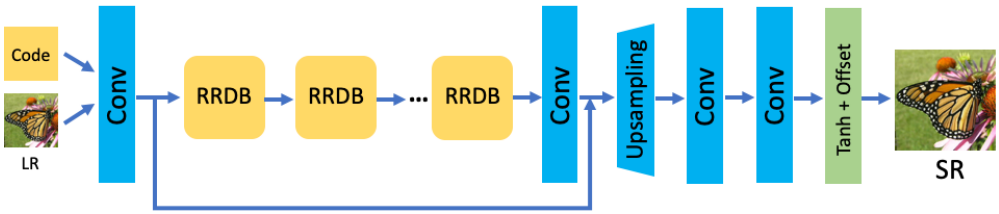


Figure 3.11: Network architecture for our super-resolution model.

Single Image Super-Resolution The network architecture is based on residual-in-residual dense network proposed by [124]. We add an additional input for the latent vector $\mathbf{z}$, which is concatenated with the low-resolution input image before being fed into the first layer. In addition, we add a tanh activation and an offset after the last layer to ensure the output lies in the range $[0, 1]$. The network architecture is shown in Figure 3.11. For the distance metric used in the loss function, we use the ℓ_2 norm (i.e. $p = 2$) and the raw pixel values and the relu5_4 activations in VGG-19 [115] as the features $\Phi_i(\cdot)$'s.

We first pretrain the network with the latent code input being set to zero, and subsequently train it on random latent code input. We use nearest neighbour interpolation for the upsampling layer initially and switch to bilinear upsampling later on.

Image Synthesis from Scene Layout The network architecture is based on the cascaded refinement network (CRN) architecture proposed in [34]. Instead of having multiple groups of three output channels, each of which is supposed to capture a different mode, we only use a single group of three output channels. To endow the network with the capability to generate an arbitrary number of modes, we add additional input channels for the latent code $\mathbf{z}$. This new model can be then interpreted as an implicit probabilistic model, which we can

train using conditional IMLE.

Because the input segmentation maps are provided at a high resolution, a noise input $\mathbf{z}$ of size $h \times w \times d$ could be very high-dimensional. To improve sample efficiency, we force the noise to lie on a low-dimensional manifold. To this end, we add a noise encoder module, which is a 3-layer convolutional network that takes $\hat{L}$ and noise sampled from a Gaussian as input and outputs an encoded noise vector $\hat{\mathbf{z}}$ of size $h \times w \times d$. We replace the original $\mathbf{z}$ with the encoded $\hat{\mathbf{z}}$ and leave the rest of the architecture unchanged. For the distance metric used in the loss function, we use the ℓ_1 norm (i.e. $p = 1$) and the conv1_2, conv2_2, conv3_2, conv4_2, and conv5_2 activations in VGG-19 [115] as the features $\Phi_i(\cdot)$'s.

Even for diverse datasets like GTA-5, there can be a strong bias towards objects with relatively common appearance. For example, black cars can be much more common than red cars, and so if we were to train a model naïvely, we would need to generate a lot of samples before seeing a red car. To address this, we propose two rebalancing strategies.

We first rebalance the dataset to increase the chance of rare images being sampled when populating S (as shown in Algorithm 7). To this end, for each image in the training set, we calculate the average pixel vector of each semantic class in that image. More concretely, we compute the following for each image k:

$$C^k(p) = \frac{\sum_{i=1}^{h} \sum_{j=1}^{w} L^k_{i,j,p} I^k_{i,j}}{\sum_{i=1}^{h} \sum_{j=1}^{w} L^k_{i,j,p}}$$

For each category p, we consider the set of average pixel vectors for that category in all training images, i.e.: $\{C^k(p) | k \in \{1, \ldots, N\}$ such that class p appears in $L^k\}$. We then fit a Gaussian kernel density estimate to this set and obtain an estimate of the distribution of average pixels of category p. Let $D_p(\cdot)$ denote the estimated probability density function (PDF) for category p. Given the k-th training example, we define the *rarity score* of category p:

$$R_p(k) = \begin{cases} \frac{1}{D_p(C^k(p))} & \text{class } p \text{ appears in } L^k \\ 0 & \text{otherwise} \end{cases}$$

We allocate a portion of the batch S in Algorithm 7 to each of the top five categories that have the largest overall area across the dataset. For each category, we sample training images based on the rarity score and effectively upweight images containing objects with rare appearance. The rationale for selecting the categories with the largest areas is because they tend to appear more frequently and be more visually prominent. If we were to allocate a fixed portion of the batch to rare categories, we would risk overfitting to images containing those categories.

We then rebalance the pixels within the same training image, because it can contain both common and rare objects. Therefore, we modify the loss function so that the objects with rare appearance are upweighted. For each training example (L^i, I^i), we define a rarity score mask $\mathcal{M}^i \in \mathbb{R}^{h \times w \times 1}$:

$$\mathcal{M}^i_{j,k} = R_p(i) \quad \text{if pixel } (j, k) \text{ belongs to class } p$$

σ	BicycleGAN	SRIM
0.3	4.31×10^{-2}	$\mathbf{5.91 \times 10^{-2}}$
0.2	5.19×10^{-3}	$\mathbf{9.23 \times 10^{-3}}$
0.15	4.67×10^{-4}	$\mathbf{1.02 \times 10^{-3}}$

Table 3.2: Comparison of faithfulness-weighted variance achieved by the proposed method (SRIM) and the leading multimodal image synthesis method, BicycleGAN. Higher value means richer variation among the generated samples.

We then normalize $\mathcal{M}^i$ so that every entry lies in $(0, 1]$:

$$\hat{\mathcal{M}}^i = \mathcal{M}^i / \max_{j,k} \mathcal{M}^i_{j,k}$$

The mask is then applied to the loss function (3.1) and the new loss $\hat{\mathcal{L}}$ becomes:

$$\hat{\mathcal{L}} = \sum_{i=1}^{l} \lambda_l ||\hat{\mathcal{M}}_l \circ \left[\Phi_l(I) - \Phi_l(\widetilde{I})\right]||_1$$

Here $\hat{\mathcal{M}}_l$ is the rarity score mask downsampled to match the size of $\Phi_l(\cdot)$

3.7.7 Results

Single Image Super-Resolution We would like a multimodal method to produce samples that are both diverse and consistent with the low-resolution input image. We propose a evaluation metric, which we call faithfulness-weighted variance, that captures both of these properties. It is defined as follows:

$$\mathcal{M} = \sum_i \sum_j w_i d(\tilde{\mathbf{y}}_{i,j}, \bar{\mathbf{y}}_j), \text{where } w_i = \exp\left(\frac{-d(\tilde{\mathbf{y}}_{i,j}, \mathbf{y}_j)}{2\sigma^2}\right)$$

where $\tilde{\mathbf{y}}_{i,j}$ is the i^{th} generated sample with respect to the j^{th} input. $\bar{\mathbf{y}}_j$ is the mean sample generated for j^{th} input image. The w_i is the faithfulness coefficient which takes the value of a Gaussian kernel evaluated on the perceptual distance between a generated sample and the true image, as measured by the LPIPS metric [128], which is denoted by $d(\cdot, \cdot)$. We report results on a range of different σ's for the Gaussian kernel. Intuitively, the faithfulness coefficient ensures that a sample only contributes significantly to the metric if it is perceptually consistent with the true image.

We evaluated the proposed method (SRIM) and BicycleGAN on a test set of 80 unseen images. On each image, 50 samples are generated using each method. The average faithfulness-weighted variance for both methods are reported in Table 3.2. It can be seen that SRIM achieved a higher weighted variance than BicycleGAN for all values of σ, which

(a) Pix2pix-HD+noise (b) BicycleGAN

(c) CRN (d) Our model

Figure 3.12: Comparison of different image samples generated from the same input scene layout. The bottom-left image in (a) is the input scene layout and we generate 9 samples for each model.

indicates it is capable of generating more diverse results that are also perceptually consistent with the true image. This is also reflected in the qualitative results, as shown in Figure 3.9.

Because there has been no prior work on multimodal super-resolution, we compare to the leading generic multimodal image synthesis method, BicycleGAN [130]. To cast super-resolution into the image-to-image translation framework, which requires the input and output resolutions to be the same, we first use bicubic upsampling to scale the 32×32 input image to 256×256, and then use the upscaled image as the input to BicyleGAN. We used the official implementation and trained for 800 epochs according to the recommended settings.

Image Synthesis from Scene Layout We train our model on 12403 training images and evaluate on the validation set of 6383 images. Each image has a resolution of 256×512. We add 10 noise channels and set the hyperparameters shown in Algorithm 7 to the following values: $|S| = 400$, $m = 10$, $M = 10000$, $|\hat{S}| = 1$ and $\eta = 1e - 5$.

We measure the diversity of each method by generating 40 pairs of output images for each of 100 input scene layouts from the test set. We then compute the average distance between each pair of output images for each given input scene layout, which is then averaged over all input scene layouts. The distance metric we use is LPIPS [128], which is designed

(a) Our model w/o the noise encoder and rebalancing scheme

(b) Our model w/o the noise encoder

(c) Our model w/o the rebalancing scheme

(d) Our model

Figure 3.13: Ablation study using the same input scene layout as in Fig. 3.12.

to measure perceptual dissimilarity. The results are shown in Table 3.3. As shown, the proposed method outperforms the baselines by a large margin. We also perform an ablation study and find that the proposed method performs better than variants that remove the noise encoder or the rebalancing scheme, which demonstrates the value of each component of our method.

We also evaluate the generated image quality by human evaluation. Since it is difficult for humans to compare images with different styles, we selected the images that are closest to the ground truth image in ℓ_1 distance among the images generated by CRN and our method. We then asked 62 human subjects to evaluate the images generated for 20 scene layouts. For each scene layout, they were asked to compare the image generated by CRN to the image generated by our method and judge which image exhibited more obvious synthetic patterns. The result is shown in Table 3.4.

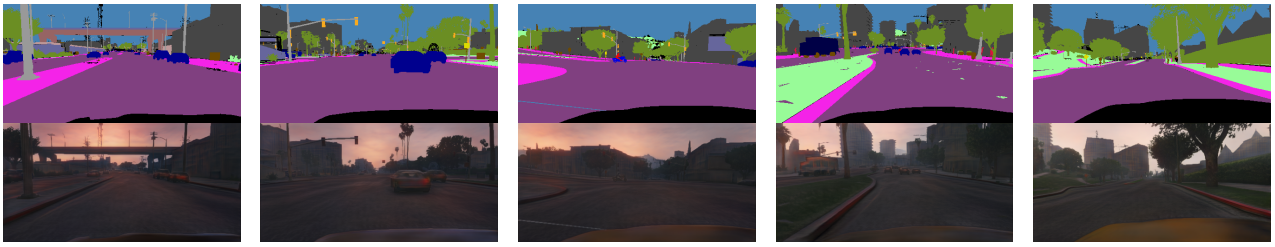
A qualitative comparison is shown in Fig. 3.12. We compare to three baselines, BicycleGAN [131], Pix2pix-HD with latent noise input [123] and CRN. As shown, Pix2pix-HD generates almost identical images, BicycleGAN generates images with heavy distortions and CRN generates images with little diversity. In comparison, the images generated by our method are diverse and do not suffer from distortions. We also perform an ablation study in Fig. 3.13, which shows that each component of our method is important.

In addition, we perform linear interpolation of noise vectors to evaluate the quality of

(a) Change from daytime to night time

(b) Change of car colors

Figure 3.14: Images generated by interpolating between latent noise vectors.



(a) (b) (c) (d) (e)

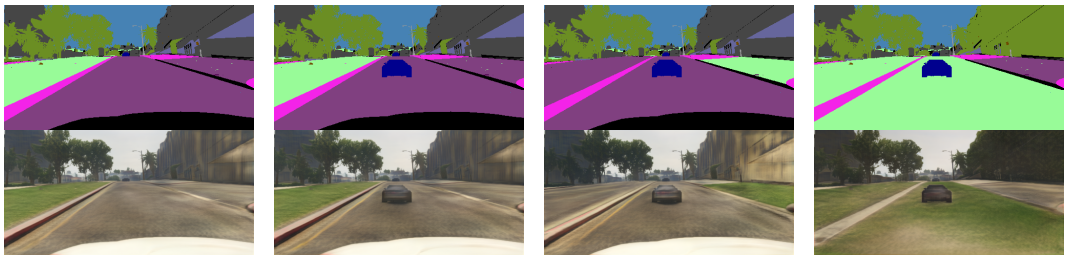
Figure 3.15: Style consistency with the same latent noise vector. (a) is the original input-output pair. We use the same latent noise vector used in (a) and apply it to (b),(c),(d) and (e)

Model	LPIPS score
CRN	0.11
CRN+noise	0.12
Ours w/o noise encoder	0.10
Ours w/o rebalancing scheme	0.17
Ours	**0.19**

Table 3.3: LPIPS score. We show the average perceptual distance of different models (including ablation study) and our proposed model exhibited the greatest diversity.

	% of Images Containing More Artifacts
CRN	0.636 ± 0.242
Our method	$\mathbf{0.364} \pm 0.242$

Table 3.4: Average percentage of images that are judged by humans to exhibit more obvious synthetic patterns. Lower is better.



(a) Original (b) Add Car (c) Change Road to Grass (d) Change Building to Trees

Figure 3.16: Generated images on four input scene layouts (which were obtained by manual editing). For each generated image, the same latent noise vector was used. (a) is the original input semantic map and the generated output, (b) adds a car on the road, (c) changes the grass on the left to road and change the side walk on the right to grass and (d) changes the building on the right to tree and changes all road to grass.

the learned latent space of noise vectors. As shown in 3.14(a), by interpolating between the noise vectors corresponding to generated images during daytime and nighttime respectively, we obtain a smooth transition from daytime to nighttime. We also show the transition in car colour in 3.14(b). This suggests that the learned latent space is sensible and captures the variation along both the daytime-nightime axis and the colour axis.

Finally, a successful method for image synthesis from scene layouts enables users to manually edit the semantic map to synthesize desired imagery. One can do this simply by adding/deleting objects or changing the class label of a certain object. In Figure 3.16 we show several such changes. Note that all four inputs use the same latent noise vector; as shown, the images are highly consistent in terms of style, which is quite useful because the style should remain the same after editing the layout. We further demonstrate this in Figure 3.15 where we apply the latent noise vector used in (a) to vastly different segmentation maps in (b),(c),(d),(e) and the sunset style is preserved across the different segmentation maps.

Appendix A

Nearest Neighbour Search

Below, we present proofs of the results shown in the main part of the dissertation. Throughout our proofs, we use $\{p^{(i)}\}_{i=1}^n$ to denote a re-ordering of the points $\{p^i\}_{i=1}^n$ so that $p^{(i)}$ is the i^{th} closest point to the query q. For any given projection direction u_{jl} associated with a simple index, we also consider a ranking of the points $\{p^i\}_{i=1}^n$ by their distance to q under projection u_{jl} in nondecreasing order. We say points are ranked before others if they appear earlier in this ranking.

A.1 Generalized Union Bound

Lemma 1. *For any set of events $\{E_i\}_{i=1}^N$, the probability that at least k' of them happen is at most $\frac{1}{k'}\sum_{i=1}^N \Pr(E_i)$.*

Proof. For any set $T \subseteq [N]$, define $\tilde{E}_T$ to be the intersection of events indexed by T and complements of events not indexed by T, i.e. $\tilde{E}_T = \left(\bigcap_{i\in T} E_i\right) \cap \left(\bigcap_{i\notin T} \overline{E}_i\right)$. Observe that $\left\{\tilde{E}_T\right\}_{T\subseteq[N]}$ are disjoint and that for any $I \subseteq [N]$, $\bigcap_{i\in I} E_i = \bigcup_{T\supseteq I} \tilde{E}_T$. The event that at least k' of E_i's happen is $\bigcup_{I\subseteq[N]:|I|=k'} \bigcap_{i\in I} E_i$, which is equivalent to $\bigcup_{I\subseteq[N]:|I|=k'} \bigcup_{T\supseteq I} \tilde{E}_T = \bigcup_{T\subseteq[N]:|T|\geq k'} \tilde{E}_T$. We will henceforth use $\mathcal{T}$ to denote $\{T \subseteq [N] : |T| \geq k'\}$. Since $\mathcal{T}$ is a finite set, we can impose an ordering on its elements and denote the l^{th} element as T_l. The event can therefore be rewritten as $\bigcup_{l=1}^{|\mathcal{T}|} \tilde{E}_{T_l}$.

Define $E'_{i,j}$ to be $E_i \setminus \left(\bigcup_{l=j+1}^{|\mathcal{T}|} \tilde{E}_{T_l}\right)$. We claim that $\sum_{i=1}^N \Pr\left(E'_{i,j}\right) \geq k' \sum_{l=1}^j \Pr\left(\tilde{E}_{T_l}\right)$ for all $j \in \{0, \ldots, |\mathcal{T}|\}$. We will show this by induction on j.

For $j = 0$, the claim is vacuously true because probabilities are non-negative. For $j > 0$, we observe that $E'_{i,j} = \left(E'_{i,j} \setminus \tilde{E}_{T_j}\right) \cup \left(E'_{i,j} \cap \tilde{E}_{T_j}\right) = E'_{i,j-1} \cup \left(E'_{i,j} \cap \tilde{E}_{T_j}\right)$ for all i. Since $E'_{i,j} \setminus \tilde{E}_{T_j}$ and $E'_{i,j} \cap \tilde{E}_{T_j}$ are disjoint, $\Pr\left(E'_{i,j}\right) = \Pr\left(E'_{i,j-1}\right) + \Pr\left(E'_{i,j} \cap \tilde{E}_{T_j}\right)$.

Consider the quantity $\sum_{i \in T_j} \Pr\left(E'_{i,j}\right)$, which is $\sum_{i \in T_j} \left(\Pr\left(E'_{i,j-1}\right) + \Pr\left(E'_{i,j} \cap \tilde{E}_{T_j}\right)\right)$ by the above observation. For each $i \in T_j, \tilde{E}_{T_j} \subseteq E_i$, and so $\tilde{E}_{T_j} \setminus \left(\bigcup_{l=j+1}^{|\mathcal{T}|} \tilde{E}_{T_l}\right) \subseteq E_i \setminus \left(\bigcup_{l=j+1}^{|\mathcal{T}|} \tilde{E}_{T_l}\right) = E'_{i,j}$. Because $\left\{\tilde{E}_{T_l}\right\}_{l=j}^{|\mathcal{T}|}$ are disjoint, $\tilde{E}_{T_j} \setminus \left(\bigcup_{l=j+1}^{|\mathcal{T}|} \tilde{E}_{T_l}\right) = \tilde{E}_{T_j}$. Hence, $\tilde{E}_{T_j} \subseteq E'_{i,j}$ and so $E'_{i,j} \cap \tilde{E}_{T_j} = \tilde{E}_{T_j}$. Thus, $\sum_{i \in T_j} \Pr\left(E'_{i,j}\right) = |T_j| \Pr\left(\tilde{E}_{T_j}\right) + \sum_{i \in T_j} \Pr\left(E'_{i,j-1}\right)$.

It follows that $\sum_{i=1}^{N} \Pr\left(E'_{i,j}\right) = |T_j| \Pr\left(\tilde{E}_{T_j}\right) + \sum_{i \in T_j} \Pr\left(E'_{i,j-1}\right) + \sum_{i \notin T_j} \Pr\left(E'_{i,j}\right)$. Because $\Pr\left(E'_{i,j}\right) = \Pr\left(E'_{i,j-1}\right) + \Pr\left(E'_{i,j} \cap \tilde{E}_{T_j}\right) \geq \Pr\left(E'_{i,j-1}\right)$ and $|T_j| \geq k'$, $\sum_{i=1}^{N} \Pr\left(E'_{i,j}\right) \geq k' \Pr\left(\tilde{E}_{T_j}\right) + \sum_{i=1}^{N} \Pr\left(E'_{i,j-1}\right)$. By the inductive hypothesis, $\sum_{i=1}^{N} \Pr\left(E'_{i,j-1}\right) \geq k' \sum_{l=1}^{j-1} \Pr\left(\tilde{E}_{T_l}\right)$. Therefore, $\sum_{i=1}^{N} \Pr\left(E'_{i,j}\right) \geq k' \sum_{l=1}^{j} \Pr\left(\tilde{E}_{T_l}\right)$, which concludes the induction argument.

The lemma is a special case of this claim when $j = |\mathcal{T}|$, since $E'_{i,|\mathcal{T}|} = E_i$ and $\sum_{l=1}^{|\mathcal{T}|} \Pr\left(\tilde{E}_{T_l}\right) = \Pr\left(\bigcup_{l=1}^{|\mathcal{T}|} \tilde{E}_{T_l}\right)$.

□

A.2 Standard DCI

Below are the proofs of the results used to show the complexities of standard DCI.

Lemma 2. *Let $v^l, v^s \in \mathbb{R}^d$ such that $\left\|v^l\right\|_2 > \left\|v^s\right\|_2$, and $u \in \mathbb{R}^d$ be a unit vector drawn uniformly at random. Then the probability of v^s being at least as long as v^l under projection u is at most $1 - \frac{2}{\pi} \cos^{-1}\left(\left\|v^s\right\|_2 / \left\|v^l\right\|_2\right)$.*

Proof. Assuming that v^l and v^s are not collinear, consider the two-dimensional subspace spanned by v^l and v^s, which we will denote as P. (If v^l and v^s are collinear, we define P to be the subspace spanned by v^l and an arbitrary vector that's linearly independent of v^l.) For any vector w, we use $w^{\parallel}$ and $w^{\perp}$ to denote the components of w in P and $P^{\perp}$ such that $w = w^{\parallel} + w^{\perp}$. For $w \in \{v^s, v^l\}$, because $w^{\perp} = 0$, $\langle w, u \rangle = \langle w, u^{\parallel} \rangle$. So, we can limit our attention to P for this analysis. We parameterize $u^{\parallel}$ in terms of its angle relative to v^l, which we denote as θ. Also, we denote the angle of $u^{\parallel}$ relative to v^s as ψ. Then,

$$
\begin{aligned}
&\Pr\left(\left|\langle v^l, u\rangle\right| \leq \left|\langle v^s, u\rangle\right|\right) \\
&= \Pr\left(\left|\langle v^l, u^{\parallel}\rangle\right| \leq \left|\langle v^s, u^{\parallel}\rangle\right|\right) \\
&= \Pr\left(\left\|v^l\right\|_2 \left\|u^{\parallel}\right\|_2 |\cos\theta| \leq \left\|v^s\right\|_2 \left\|u^{\parallel}\right\|_2 |\cos\psi|\right) \\
&\leq \Pr\left(|\cos\theta| \leq \frac{\|v^s\|_2}{\|v^l\|_2}\right) \\
&= 2\Pr\left(\theta \in \left[\cos^{-1}\left(\frac{\|v^s\|_2}{\|v^l\|_2}\right), \pi - \cos^{-1}\left(\frac{\|v^s\|_2}{\|v^l\|_2}\right)\right]\right) \\
&= 1 - \frac{2}{\pi}\cos^{-1}\left(\frac{\|v^s\|_2}{\|v^l\|_2}\right)
\end{aligned}
$$

□

Lemma 3. *Let $\left\{v_i^l\right\}_{i=1}^N$ be a set of vectors such that $\left\|v_i^l\right\|_2 > \left\|v^s\right\|_2 \ \forall i \in [N]$. Then the probability that there is a subset of k' vectors from $\left\{v_i^l\right\}_{i=1}^N$ that are all not longer than v^s under projection is at most $\frac{1}{k'}\sum_{i=1}^N \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v^s\right\|_2 / \left\|v_i^l\right\|_2\right)\right)$. Furthermore, if $k' = N$, this probability is at most $\min_{i\in[N]}\left\{1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v^s\right\|_2 / \left\|v_i^l\right\|_2\right)\right\}$.*

Proof. For a given subset $I \subseteq [N]$ of size k', the probability that all vectors indexed by elements in I are not longer than v^s under projection u is at most $\Pr\left(u \in \bigcap_{i\in I} U(v^s, v_i^l)\right) = \text{area}\left(\bigcap_{i\in I} U(v^s, v_i^l)\right)/\text{area}\,(B)$. So, the probability that this occurs on some subset I is at most $\Pr\left(u \in \bigcup_{I\subseteq[N]:|I|=k'} \bigcap_{i\in I} U(v^s, v_i^l)\right) = \text{area}\left(\bigcup_{I\subseteq[N]:|I|=k'} \bigcap_{i\in I} U(v^s, v_i^l)\right)/\text{area}\,(B)$.

Observe that each point in $\bigcup_{I\subseteq[N]:|I|=k'} \bigcap_{i\in I} U(v^s, v_i^l)$ must be covered by at least k' $U(v^s, v_i^l)$'s. So,

$$
k' \cdot \text{area}\left(\bigcup_{I\subseteq[N]:|I|=k'} \bigcap_{i\in I} U(v^s, v_i^l)\right) \leq \sum_{i=1}^N \text{area}\left(U(v^s, v_i^l)\right)
$$

It follows that the probability this event occurs on some subset I is bounded above by $\frac{1}{k'}\sum_{i=1}^N \frac{\text{area}\left(U(v^s, v_i^l)\right)}{\text{area}(B)} = \frac{1}{k'}\sum_{i=1}^N \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v^s\right\|_2 / \left\|v_i^l\right\|_2\right)\right)$.

If $k' = N$, we use the fact that $\text{area}\left(\bigcap_{i\in[N]} U(v^s, v_i^l)\right) \leq \min_{i\in[N]}\left\{\text{area}\left(U(v^s, v_i^l)\right)\right\}$ to obtain the desired result. □

Theorem 1. *Let $\left\{v_i^l\right\}_{i=1}^N$ and $\left\{v_{i'}^s\right\}_{i'=1}^{N'}$ be sets of vectors such that $\left\|v_i^l\right\|_2 > \left\|v_{i'}^s\right\|_2 \forall i \in [N], i' \in [N']$. Then the probability that there is a subset of k' vectors from $\left\{v_i^l\right\}_{i=1}^N$ that are all not longer than some $v_{i'}^s$ under projection is at most $\frac{1}{k'}\sum_{i=1}^N \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v_{\max}^s\right\|_2 / \left\|v_i^l\right\|_2\right)\right)$, where $\left\|v_{\max}^s\right\|_2 \geq \left\|v_{i'}^s\right\|_2 \forall i' \in [N']$.*

Proof. The probability that this event occurs is at most
$\Pr\left(u \in \bigcup_{i'\in[N']} \bigcup_{I\subseteq[N]:|I|=k'} \bigcap_{i\in I} U(v_{i'}^s, v_i^l)\right)$. We observe that for all i, i',

$\left\{\theta \left| |\cos\theta| \leq \|v^s_{i'}\|_2 / \|v^l_i\|_2 \right.\right\} \subseteq \left\{\theta \left| |\cos\theta| \leq \|v^s_{\max}\|_2 / \|v^l_i\|_2 \right.\right\}$, which implies that $U(v^s_{i'}, v^l_i) \subseteq U(v^s_{\max}, v^l_i)$.

If we take the intersection followed by union on both sides, we obtain $\bigcup_{I\subseteq[N]:|I|=k'} \bigcap_{i\in I} U(v^s_{i'}, v^l_i) \subseteq \bigcup_{I\subseteq[N]:|I|=k'} \bigcap_{i\in I} U(v^s_{\max}, v^l_i)$. Because this is true for all i', $\bigcup_{i'\in[N']} \bigcup_{I\subseteq[N]:|I|=k'} \bigcap_{i\in I} U(v^s_{i'}, v^l_i) \subseteq \bigcup_{I\subseteq[N]:|I|=k'} \bigcap_{i\in I} U(v^s_{\max}, v^l_i)$.

Therefore, this probability is bounded above by $\Pr\left(u \in \bigcup_{I\subseteq[N]:|I|=k'} \bigcap_{i\in I} U(v^s_{\max}, v^l_i)\right)$. By Lemma 3, this is at most $\frac{1}{k'} \sum_{i=1}^{N} \left(1 - \frac{2}{\pi}\cos^{-1}\left(\|v^s_{\max}\|_2 / \|v^l_i\|_2\right)\right)$.

□

Next we prove two intermediate results.

Lemma 11. *The probability that for all constituent simple indices of a composite index, fewer than n_0 points exist that are not the true k-nearest neighbours but are ranked before some of them, is at least* $\left[1 - \frac{1}{n_0-k} \sum_{i=2k+1}^{n} \frac{\|p^{(k)}-q\|_2}{\|p^{(i)}-q\|_2}\right]^m$.

Proof. For any given simple index, we will refer to the points that are not the true k-nearest neighbours but are ranked before some of them as *extraneous points*. We furthermore categorize the extraneous points as either *reasonable* or *silly*. An extraneous point is reasonable if it is one of the $2k$-nearest neighbours, and is silly otherwise. Since there can be at most k reasonable extraneous points, there must be at least $n_0 - k$ silly extraneous points. Therefore, the event that n_0 extraneous points exist must be contained in the event that $n_0 - k$ silly extraneous points exist.

We find the probability that such a set of silly extraneous points exists for any given simple index. By Theorem 1, where we take $\{v^s_{i'}\}_{i'=1}^{N'}$ to be $\{p^{(i)} - q\}_{i=1}^{k}$, $\{v^l_i\}_{i=1}^{N}$ to be $\{p^{(i)} - q\}_{i=2k+1}^{n}$ and k' to be $n_0 - k$, the probability that there are at least $n_0 - k$ silly extraneous points is at most $\frac{1}{n_0-k} \sum_{i=2k+1}^{n} \left(1 - \frac{2}{\pi}\cos^{-1}\left(\frac{\|p^{(k)}-q\|_2}{\|p^{(i)}-q\|_2}\right)\right)$. This implies that the probability that at least n_0 extraneous points exist is bounded above by the same quantity, and so the probability that fewer than n_0 extraneous points exist is at least $1 - \frac{1}{n_0-k} \sum_{i=2k+1}^{n} \left(1 - \frac{2}{\pi}\cos^{-1}\left(\frac{\|p^{(k)}-q\|_2}{\|p^{(i)}-q\|_2}\right)\right)$. Hence, the probability that fewer than n_0 extraneous points exist for all constituent simples indices of a composite index is at least $\left[1 - \frac{1}{n_0-k} \sum_{i=2k+1}^{n} \left(1 - \frac{2}{\pi}\cos^{-1}\left(\frac{\|p^{(k)}-q\|_2}{\|p^{(i)}-q\|_2}\right)\right)\right]^m$. Using the fact that $1 - (2/\pi)\cos^{-1}(x) \leq x \; \forall x \in [0,1]$, this quantity is at least $\left[1 - \frac{1}{n_0-k} \sum_{i=2k+1}^{n} \frac{\|p^{(k)}-q\|_2}{\|p^{(i)}-q\|_2}\right]^m$. □

Lemma 12. *On a dataset with global relative sparsity (k, γ), the probability that for all constituent simple indices of a composite index, fewer than n_0 points exist that are not the true k-nearest neighbours but are ranked before some of them, is at least* $\left[1 - \frac{1}{n_0-k} O\left(\max(k\log(n/k), k(n/k)^{1-\log_2\gamma})\right)\right]^m$.

Proof. By definition of global relative sparsity, for all $i \geq 2k+1$, $\left\|p^{(i)} - q\right\|_2 > \gamma \left\|p^{(k)} - q\right\|_2$. By applying this recursively, we see that for all $i \geq 2^{i'}k + 1$, $\left\|p^{(i)} - q\right\|_2 > \gamma^{i'} \left\|p^{(k)} - q\right\|_2$. It follows that $\sum_{i=2k+1}^{n} \frac{\left\|p^{(k)}-q\right\|_2}{\left\|p^{(i)}-q\right\|_2}$ is less than $\sum_{i'=1}^{\lceil \log_2(n/k) \rceil - 1} 2^{i'} k \gamma^{-i'}$. If $\gamma \geq 2$, this quantity is at most $k \log_2\left(\frac{n}{k}\right)$. If $1 \leq \gamma < 2$, this quantity is:

$$
\begin{aligned}
& k\left(\frac{2}{\gamma}\right)\left(\left(\frac{2}{\gamma}\right)^{\lceil \log_2(n/k) \rceil - 1} - 1\right) / \left(\frac{2}{\gamma} - 1\right) \\
&= O\left(k\left(\frac{2}{\gamma}\right)^{\lceil \log_2(n/k) \rceil - 1}\right) \\
&= O\left(k\left(\frac{n}{k}\right)^{1-\log_2 \gamma}\right)
\end{aligned}
$$

Combining this bound with Lemma 11 yields the desired result.

□

Lemma 4. *For a dataset with global relative sparsity (k, γ), there is some $\tilde{k} \in \Omega(\max(k \log(n/k), k(n/k)^{1-\log_2 \gamma}))$ such that the probability that the candidate points retrieved from a given composite index do not include some of the true k-nearest neighbours is at most some constant $\alpha < 1$.*

Proof. We will refer to points ranked in the top $\tilde{k}$ positions that are the true k-nearest neighbours as *true positives* and those that are not as *false positives*. Additionally, we will refer to points not ranked in the top $\tilde{k}$ positions that are the true k-nearest neighbours as *false negatives*.

When not all the true k-nearest neighbours are in the top $\tilde{k}$ positions, then there must be at least one false negative. Since there are at most $k-1$ true positives, there must be at least $\tilde{k} - (k-1)$ false positives.

Since false positives are not the true k-nearest neighbours but are ranked before the false negative, which is a true k-nearest neighbour, we can apply Lemma 12. By taking n_0 to be $\tilde{k} - (k-1)$, we obtain a lower bound on the probability of the existence of fewer than $\tilde{k} - (k-1)$ false positives for all constituent simple indices of the composite index, which is $\left[1 - \frac{1}{\tilde{k}-2k+1} O\left(\max(k \log(n/k), k(n/k)^{1-\log_2 \gamma})\right)\right]^m$. If each simple index has fewer than $\tilde{k} - (k-1)$ false positives, then the top $\tilde{k}$ positions must contain all the true k-nearest neighbours. Since this is true for all constituent simple indices, all the true k-nearest neighbours must be among the candidate points after $\tilde{k}$ iterations of the outer loop. The failure probability is therefore at most $1 - \left[1 - \frac{1}{\tilde{k}-2k+1} O\left(\max(k \log(n/k), k(n/k)^{1-\log_2 \gamma})\right)\right]^m$. So, there is some $\tilde{k} \in \Omega(\max(k \log(n/k), k(n/k)^{1-\log_2 \gamma}))$ that makes this quantity strictly less than 1.

□

Theorem 2. *For a dataset with global relative sparsity (k, γ), for any $\epsilon > 0$, there is some L and $\tilde{k} \in \Omega(\max(k \log(n/k), k(n/k)^{1-\log_2 \gamma}))$ such that the algorithm returns the correct set of k-nearest neighbours with probability of at least $1 - \epsilon$.*

Proof. By Lemma 4, the first $\tilde{k}$ points retrieved from a given composite index do not include some of the true k-nearest neighbours with probability of at most α. For the algorithm to fail, this must occur for all composite indices. Since each composite index is constructed independently, the algorithm fails with probability of at most α^L, and so must succeed with probability of at least $1 - \alpha^L$. Since $\alpha < 1$, there is some L that makes $1 - \alpha^L \geq 1 - \epsilon$. □

Theorem 3. *The algorithm takes $O(\max(d(m + k \log(n/k), dk(n/k)^{1-1/d'})))$ time to retrieve the k-nearest neighbours at query time, where d' denotes the intrinsic dimensionality of the dataset.*

Proof. Computing projections of the query point along all u_{jl}'s takes $O(dm)$ time, since L is a constant. Searching in the binary search trees/skip lists T_{jl}'s takes $O(\log n)$ time. The total number of candidate points retrieved is at most $\Theta(\max(k \log(n/k), k(n/k)^{1-\log_2 \gamma}))$. Computing the distance between each candidate point and the query point takes at most $O(\max(dk \log(n/k), dk(n/k)^{1-\log_2 \gamma}))$ time. We can find the k closest points to q in the set of candidate points using a selection algorithm like quickselect, which takes $O(\max(k \log(n/k), k(n/k)^{1-\log_2 \gamma}))$ time on average. So, the entire algorithm takes $O(\max(d(m + k \log(n/k), dk(n/k)^{1-\log_2 \gamma})))$ time. Since $d' = 1/\log_2 \gamma$, this can be rewritten as $O(\max(d(m + k \log(n/k), dk(n/k)^{1-1/d'})))$. □

Theorem 4. *The algorithm takes $O(dn + n \log n)$ time to preprocess the data points in D at construction time.*

Proof. Computing projections of all n points along all u_{jl}'s takes $O(dn)$ time, since m and L are constants. Inserting all n points into mL self-balancing binary search trees/skip lists takes $O(n \log n)$ time. □

Theorem 5. *The algorithm requires $O(d + \log n)$ time to insert a new data point and $O(\log n)$ time to delete a data point.*

Proof. In order to insert a data point, we need to compute its projection along all u_{jl}'s and insert it into each binary search tree or skip list. Computing the projection takes $O(d)$ time and inserting an entry into a self-balancing binary search tree or skip list takes $O(\log n)$ time. In order to delete a data point, we simply remove it from each of the binary search trees or skip lists, which takes $O(\log n)$ time. □

Theorem 6. *The algorithm requires $O(n)$ space in addition to the space used to store the data.*

Proof. The only additional information that needs to be stored are the mL binary search trees or skip lists. Since n entries are stored in each binary search tree/skip list, the additional space required is $O(n)$. □

Theorem 7. *For any $\epsilon > 0$, m and L, the data-dependent algorithm returns the correct set of k-nearest neighbours of the query q with probability of at least $1 - \epsilon$.*

Proof. We analyze the probability that the algorithm fails to return the correct set of k-nearest neighbours. Let p^* denote a true k-nearest neighbour that was missed. If the algorithm fails, then for any given composite index, p^* is not among the candidate points retrieved from the said index. In other words, the composite index must have returned all these points before p^*, implying that at least one constituent simple index returns all these points before p^*. This means that all these points must appear closer to q than p^* under the projection associated with the simple index. By Lemma 3, if we take $\left\{v_i^l\right\}_{i=1}^N$ to be displacement vectors from q to the candidate points that are farther from q than p^* and v^s to be the displacement vector from q to p^*, the probability of this occurring for a given constituent simple index of the l^{th} composite index is at most $1 - \frac{2}{\pi}\cos^{-1}\left(\|p^* - q\|_2 / \|\tilde{p}_l^{\max} - q\|_2\right)$. The probability that this occurs for *some* constituent simple index is at most $1 - \left(\frac{2}{\pi}\cos^{-1}\left(\|p^* - q\|_2 / \|\tilde{p}_l^{\max} - q\|_2\right)\right)^m$. For the algorithm to fail, this must occur for all composite indices; so the failure probability is at most $\prod_{l=1}^{L}\left(1 - \left(\frac{2}{\pi}\cos^{-1}\left(\|p^* - q\|_2 / \|\tilde{p}_l^{\max} - q\|_2\right)\right)^m\right)$.

We observe that $\left\|p^* - q\right\|_2 \leq \left\|p^{(k)} - q\right\|_2 \leq \left\|\tilde{p}^{(k)} - q\right\|_2$ since there can be at most $k - 1$ points in the dataset that are closer to q than p^*. So, the failure probability can be bounded above by $\prod_{l=1}^{L}\left(1 - \left(\frac{2}{\pi}\cos^{-1}\left(\left\|\tilde{p}^{(k)} - q\right\|_2 / \left\|\tilde{p}_l^{\max} - q\right\|_2\right)\right)^m\right)$. When the algorithm terminates, we know this quantity is at most ϵ. Therefore, the algorithm returns the correct set of k-nearest neighbours with probability of at least $1 - \epsilon$. □

Theorem 8. *On a dataset with global relative sparsity (k, γ), given fixed parameters m and L, the data-dependent algorithm takes $O\left(\max\left(dk\log\left(\frac{n}{k}\right), dk\left(\frac{n}{k}\right)^{1-1/d'}, \frac{d}{\left(1-\sqrt[m]{1-\sqrt[L]{\epsilon}}\right)^{d'}}\right)\right)$ time with high probability to retrieve the k-nearest neighbours at query time, where d' denotes the intrinsic dimensionality of the dataset.*

Proof. In order to bound the running time, we bound the total number of candidate points retrieved until the stopping condition is satisfied. We divide the execution of the algorithm into two stages and analyze the algorithm's behaviour before and after it finishes retrieving all the true k-nearest neighbours. We first bound the number of candidate points the algorithm retrieves before finding the complete set of k-nearest neighbours. By Lemma 12, the probability that there exist fewer than n_0 points that are not the k-nearest neighbours but are ranked before some of them in all constituent simple indices of any given composite index is at least $\left[1 - \frac{1}{n_0 - k}O\left(\max(k\log(n/k), k(n/k)^{1-\log_2\gamma})\right)\right]^m$. We can choose some $n_0 \in \Theta\left(\max(k\log(n/k), k(n/k)^{1-\log_2\gamma})\right)$ that makes this probability arbitrarily close to 1. So, there are $\Theta\left(\max(k\log(n/k), k(n/k)^{1-\log_2\gamma})\right)$ such points in each constituent simple index with high probability, implying that the algorithm retrieves at most $\Theta\left(\max(k\log(n/k), k(n/k)^{1-\log_2\gamma})\right)$ extraneous points from any given composite index before finishing fetching all the true k-nearest neighbours. Since the number of composite indices is constant, the total number of candidate points retrieved from all composite indices during

this stage is $k + \Theta\left(\max(k\log(n/k), k(n/k)^{1-\log_2\gamma})\right) = \Theta\left(\max(k\log(n/k), k(n/k)^{1-\log_2\gamma})\right)$ with high probability.

After retrieving all the k-nearest neighbours, if the stopping condition has not yet been satisfied, the algorithm would continue retrieving points. We analyze the number of additional points the algorithm retrieves before it terminates. To this end, we bound the ratio $\left\|\tilde{p}^{(k)} - q\right\|_2 / \left\|\tilde{p}_l^{\max} - q\right\|_2$ in terms of the number of candidate points retrieved so far. Since all the true k-nearest neighbours have been retrieved, $\left\|\tilde{p}^{(k)} - q\right\|_2 = \left\|p^{(k)} - q\right\|_2$. Suppose the algorithm has already retrieved $n' - 1$ candidate points and is about to retrieve a new candidate point. Since this new candidate point must be different from any of the existing candidate points, $\left\|\tilde{p}_l^{\max} - q\right\|_2 \geq \left\|p^{(n')} - q\right\|_2$. Hence, $\left\|\tilde{p}^{(k)} - q\right\|_2 / \left\|\tilde{p}_l^{\max} - q\right\|_2 \leq \left\|p^{(k)} - q\right\|_2 / \left\|p^{(n')} - q\right\|_2$.

By definition of global relative sparsity, for all $n' \geq 2^{i'}k+1$, $\left\|p^{(n')} - q\right\|_2 > \gamma^{i'}\left\|p^{(k)} - q\right\|_2$. It follows that $\left\|p^{(k)} - q\right\|_2 / \left\|p^{(n')} - q\right\|_2 < \gamma^{-\lfloor\log_2((n'-1)/k)\rfloor}$ for all n'. By combining the above inequalities, we find an upper bound on the test statistic:

$$\begin{aligned}
&\prod_{l=1}^{L}\left(1 - \left(\frac{2}{\pi}\cos^{-1}\left(\frac{\left\|\tilde{p}^{(k)} - q\right\|_2}{\left\|\tilde{p}_l^{\max} - q\right\|_2}\right)\right)^m\right) \\
&\leq \prod_{l=1}^{L}\left(1 - \left(1 - \frac{\left\|\tilde{p}^{(k)} - q\right\|_2}{\left\|\tilde{p}_l^{\max} - q\right\|_2}\right)^m\right) \\
&< \left[1 - \left(1 - \gamma^{-\lfloor\log_2((n'-1)/k)\rfloor}\right)^m\right]^L \\
&< \left[1 - \left(1 - \gamma^{-\log_2((n'-1)/k)+1}\right)^m\right]^L
\end{aligned}$$

Hence, if $\left[1 - \left(1 - \gamma^{-\log_2((n'-1)/k)+1}\right)^m\right]^L \leq \epsilon$, then $\prod_{l=1}^{L}\left(1 - \left(\frac{2}{\pi}\cos^{-1}\left(\left\|\tilde{p}^{(k)} - q\right\|_2 / \left\|\tilde{p}_l^{\max} - q\right\|_2\right)\right)^m\right) < \epsilon$. So, for some n' that makes the former inequality true, the stopping condition would be satisfied and so the algorithm must have terminated by this point, if not earlier. By rearranging the former inequality, we find that in order for it to hold, n' must be at least $2/\left(1 - \sqrt[m]{1 - \sqrt[L]{\epsilon}}\right)^{1/\log_2\gamma}$. Therefore, the number of points the algorithm retrieves before terminating cannot exceed $2/\left(1 - \sqrt[m]{1 - \sqrt[L]{\epsilon}}\right)^{1/\log_2\gamma}$.

Combining the analysis for both stages, the number of points retrieved is at most

$$O\left(\max\left(k\log\left(\frac{n}{k}\right), k\left(\frac{n}{k}\right)^{1-\log_2\gamma}, \frac{1}{\left(1 - \sqrt[m]{1 - \sqrt[L]{\epsilon}}\right)^{\frac{1}{\log_2\gamma}}}\right)\right)$$

with high probability.

Since the time taken to compute distances between the query point and candidate points dominates, the running time is

$$O\left(\max\left(dk\log\left(\frac{n}{k}\right), dk\left(\frac{n}{k}\right)^{1-\log_2\gamma}, \frac{d}{\left(1 - \sqrt[m]{1 - \sqrt[L]{\epsilon}}\right)^{\frac{1}{\log_2\gamma}}}\right)\right)$$

with high probability.

Applying the definition of intrinsic dimensionality yields the desired result.

□

A.3 Prioritized DCI

Below are the proofs of the results used to show the complexities of Prioritized DCI.

Lemma 5. *Let $v^l, v^s \in \mathbb{R}^d$ be such that $\left\|v^l\right\|_2 > \left\|v^s\right\|_2$, $\left\{u'_j\right\}_{j=1}^M$ be i.i.d. unit vectors in $\mathbb{R}^d$ drawn uniformly at random. Then $\Pr\left(\max_j \left\{\left|\langle v^l, u'_j\rangle\right|\right\} \leq \left\|v^s\right\|_2\right) = \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v^s\right\|_2 / \left\|v^l\right\|_2\right)\right)^M$.*

Proof. The event $\left\{\max_j \left\{\left|\langle v^l, u'_j\rangle\right|\right\} \leq \|v^s\|_2\right\}$ is equivalent to the event that $\left\{\left|\langle v^l, u'_j\rangle\right| \leq \|v^s\|_2 \ \forall j\right\}$, which is the intersection of the events $\left\{\left|\langle v^l, u'_j\rangle\right| \leq \|v^s\|_2\right\}$. Because u'_j's are drawn independently, these events are independent.

Let θ_j be the angle between v^l and u'_j, so that $\langle v^l, u'_j\rangle = \left\|v^l\right\|_2 \cos\theta_j$. Since u'_j is drawn uniformly, θ_j is uniformly distributed on $[0, 2\pi]$. Hence,

$$
\begin{aligned}
&\Pr\left(\max_j \left\{\left|\langle v^l, u'_j\rangle\right|\right\} \leq \|v^s\|_2\right) \\
&= \prod_{j=1}^M \Pr\left(\left|\langle v^l, u'_j\rangle\right| \leq \|v^s\|_2\right) \\
&= \prod_{j=1}^M \Pr\left(|\cos\theta_j| \leq \frac{\|v^s\|_2}{\|v^l\|_2}\right) \\
&= \prod_{j=1}^M \left(2\Pr\left(\theta_j \in \left[\cos^{-1}\left(\frac{\|v^s\|_2}{\|v^l\|_2}\right), \pi - \cos^{-1}\left(\frac{\|v^s\|_2}{\|v^l\|_2}\right)\right]\right)\right) \\
&= \left(1 - \frac{2}{\pi}\cos^{-1}\left(\frac{\|v^s\|_2}{\|v^l\|_2}\right)\right)^M
\end{aligned}
$$

□

Theorem 9. *Let $\left\{v^l_i\right\}_{i=1}^N$ and $\left\{v^s_{i'}\right\}_{i'=1}^{N'}$ be sets of vectors such that $\left\|v^l_i\right\|_2 > \left\|v^s_{i'}\right\|_2 \ \forall i \in [N], i' \in [N']$. Furthermore, let $\left\{u'_{ij}\right\}_{i\in[N], j\in[M]}$ be random uniformly distributed unit vectors such that $u'_{i1}, \ldots, u'_{iM}$ are independent for any given i. Consider the events $\left\{\exists v^s_{i'} \text{ s.t. } \max_j \left\{\left|\langle v^l_i, u'_{ij}\rangle\right|\right\} \leq \left\|v^s_{i'}\right\|_2\right\}_{i=1}^N$. The probability that at least k' of these events occur is at most $\frac{1}{k'}\sum_{i=1}^N \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v^s_{\max}\right\|_2 / \left\|v^l_i\right\|_2\right)\right)^M$, where $\left\|v^s_{\max}\right\|_2 = \max_{i'}\left\{\left\|v^s_{i'}\right\|_2\right\}$. Furthermore, if $k' = N$, it is at most $\min_{i\in[N]}\left\{\left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v^s_{\max}\right\|_2 / \left\|v^l_i\right\|_2\right)\right)^M\right\}$.*

Proof. The event that $\exists v^s_{i'}$ s.t. $\max_j \left\{\left|\langle v^l_i, u'_{ij}\rangle\right|\right\} \leq \left\|v^s_{i'}\right\|_2$ is equivalent to the event that $\max_j \left\{\left|\langle v^l_i, u'_{ij}\rangle\right|\right\} \leq \max_{i'}\left\{\left\|v^s_{i'}\right\|_2\right\} = \left\|v^s_{\max}\right\|_2$. Take E_i to be the event that $\max_j \left\{\left|\langle v^l_i, u'_{ij}\rangle\right|\right\} \leq \left\|v^s_{\max}\right\|_2$. By Lemma 5, $\Pr(E_i) \leq \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v^s_{\max}\right\|_2 / \left\|v^l_i\right\|_2\right)\right)^M$. It follows from Lemma 1

that the probability that k' of E_i's occur is at most $\frac{1}{k'}\sum_{i=1}^{N}\Pr(E_i) \leq \frac{1}{k'}\sum_{i=1}^{N}\left(1-\frac{2}{\pi}\cos^{-1}\left(\left\|v_{\max}^{s}\right\|_2 / \left\|v_i^l\right\|_2\right)\right)^M$. If $k'=N$, we use the fact that $\bigcap_{i'=1}^{N} E_{i'} \subseteq E_i \ \forall i$, which implies that $\Pr\left(\bigcap_{i'=1}^{N} E_{i'}\right) \leq \min_{i\in[N]}\Pr(E_i) \leq \min_{i\in[N]}\left\{\left(1-\frac{2}{\pi}\cos^{-1}\left(\left\|v_{\max}^{s}\right\|_2 / \left\|v_i^l\right\|_2\right)\right)^M\right\}$. □

Lemma 6. *Consider points in the order they are retrieved from a composite index that consists of m simple indices. The probability that there are at least n_0 points that are not the true k-nearest neighbours but are retrieved before some of them is at most* $\frac{1}{n_0-k}\sum_{i=2k+1}^{n}\left(1-\frac{2}{\pi}\cos^{-1}\left(\left\|p^{(k)}-q\right\|_2 / \left\|p^{(i)}-q\right\|_2\right)\right)^m$.

Proof. Points that are not the true k-nearest neighbours but are retrieved before some of them will be referred to as *extraneous points* and are divided into two categories: *reasonable* and *silly*. An extraneous point is reasonable if it is one of the $2k$-nearest neighbours, and is silly otherwise. For there to be n_0 extraneous points, there must be n_0-k silly extraneous points. Therefore, the probability that there are n_0 extraneous points is upper bounded by the probability that there are n_0-k silly extraneous points.

Since points are retrieved from the composite index in the order of increasing maximum projected distance to the query, for any pair of points p and p', if p is retrieved before p', then $\max_j\{|\langle p-q,u_{jl}\rangle|\} \leq \max_j\{|\langle p'-q,u_{jl}\rangle|\}$, where $\{u_{jl}\}_{j=1}^{m}$ are the projection directions associated with the constituent simple indices of the composite index.

By Theorem 9, if we take $\left\{v_i^l\right\}_{i=1}^{N}$ to be $\left\{p^{(i)}-q\right\}_{i=2k+1}^{n}$, $\left\{v_{i'}^s\right\}_{i'=1}^{N'}$ to be $\left\{p^{(i)}-q\right\}_{i=1}^{k}$, M to be m, $\left\{u'_{ij}\right\}_{j\in[M]}$ to be $\{u_{jl}\}_{j\in[m]}$ for all $i\in[N]$ and k' to be n_0-k, we obtain an upper bound for the probability of there being a subset of $\left\{p^{(i)}\right\}_{i=2k+1}^{n}$ of size n_0-k such that for all points p in the subset, $\max_j\{|\langle p-q,u_{jl}\rangle|\} \leq \|p'-q\|_2$ for some $p'\in\left\{p^{(i)}-q\right\}_{i=1}^{k}$. In other words, this is the probability of there being n_0-k points that are not the $2k$-nearest neighbours whose maximum projected distances are no greater than the distance from some k-nearest neighbours to the query, which is at most $\frac{1}{n_0-k}\sum_{i=2k+1}^{n}\left(1-\frac{2}{\pi}\cos^{-1}\left(\left\|p^{(k)}-q\right\|_2 / \left\|p^{(i)}-q\right\|_2\right)\right)^m$.

Since the event that $\max_j\{|\langle p-q,u_{jl}\rangle|\} \leq \max_j\{|\langle p'-q,u_{jl}\rangle|\}$ is contained in the event that $\max_j\{|\langle p-q,u_{jl}\rangle|\} \leq \|p'-q\|_2$ for any p,p', this is also an upper bound for the probability of there being n_0-k points that are not the $2k$-nearest neighbours whose maximum projected distances do not exceed those of some of the k-nearest neighbours, which by definition is the probability that there are n_0-k silly extraneous points. Since this probability is no less than the probability that there are n_0 extraneous points, the upper bound also applies to this probability. □

Lemma 7. *Consider point projections in a composite index that consists of m simple indices in the order they are visited. The probability that there are n_0 point projections that are not the true k-nearest neighbours but are visited before all true k-nearest neighbours have been retrieved is at most* $\frac{m}{n_0-mk}\sum_{i=2k+1}^{n}\left(1-\frac{2}{\pi}\cos^{-1}\left(\left\|p^{(k)}-q\right\|_2 / \left\|p^{(i)}-q\right\|_2\right)\right)$.

Proof. Projections of points that are not the true k-nearest neighbours but are visited before the k-nearest neighbours have all been retrieved will be referred to as *extraneous projections*

and are divided into two categories: *reasonable* and *silly*. An extraneous projection is reasonable if it is of one of the $2k$-nearest neighbours, and is silly otherwise. For there to be n_0 extraneous projections, there must be $n_0 - mk$ silly extraneous projections, since there could be at most mk reasonable extraneous projections. Therefore, the probability that there are n_0 extraneous projections is upper bounded by the probability that there are $n_0 - mk$ silly extraneous projections.

Since point projections are visited in the order of increasing projected distance to the query, each extraneous silly projection must be closer to the query projection than the maximum projection of some k-nearest neighbour.

By Theorem 9, if we take $\left\{v_i^l\right\}_{i=1}^N$ to be $\left\{p^{(2k+\lfloor (i-1)/m \rfloor + 1)} - q\right\}_{i=1}^{m(n-2k)}$, $\left\{v_{i'}^s\right\}_{i'=1}^{N'}$ to be $\left\{p^{(\lfloor (i-1)/m \rfloor + 1)} - q\right\}_{i=1}^{mk}$, M to be 1, $\left\{u'_{i1}\right\}_{i=1}^N$ to be $\left\{u_{(i \mod m),l}\right\}_{i=1}^{m(n-2k)}$ and k' to be $n_0 - mk$, we obtain an upper bound for the probability of there being $n_0 - mk$ point projections that are not of the $2k$-nearest neighbours whose distances to their respective query projections are no greater than the true distance between the query and some k-nearest neighbour, which is $\frac{1}{n_0 - mk} \sum_{i=2k+1}^{n} m \left(1 - \frac{2}{\pi} \cos^{-1}\left(\frac{\left\|p^{(k)} - q\right\|_2}{\left\|p^{(i)} - q\right\|_2}\right)\right)$.

Because maximum projected distances are no more than true distances, this is also an upper bound for the probability of there being $n_0 - mk$ silly extraneous projections. Since this probability is no less than the probability that there are n_0 extraneous projections, the upper bound also applies to this probability. □

Lemma 8. *On a dataset with global relative sparsity (k, γ), the quantity $\sum_{i=2k+1}^{n} \left(1 - \frac{2}{\pi} \cos^{-1}\left(\left\|p^{(k)} - q\right\|_2 / \left\|p^{(i)} - q\right\|_2\right)\right)^m$ is at most $O\left(k \max(\log(n/k), (n/k)^{1 - m \log_2 \gamma})\right)$.*

Proof. By definition of global relative sparsity, for all $i \geq 2k+1$, $\left\|p^{(i)} - q\right\|_2 > \gamma \left\|p^{(k)} - q\right\|_2$. A recursive application shows that for all $i \geq 2^{i'} k + 1$, $\left\|p^{(i)} - q\right\|_2 > \gamma^{i'} \left\|p^{(k)} - q\right\|_2$.

Applying the fact that $1 - (2/\pi)\cos^{-1}(x) \leq x \ \forall x \in [0, 1]$ and the above observation yields:

$$\sum_{i=2k+1}^{n} \left(1 - \frac{2}{\pi} \cos^{-1}\left(\frac{\left\|p^{(k)} - q\right\|_2}{\left\|p^{(i)} - q\right\|_2}\right)\right)^m$$

$$\leq \sum_{i=2k+1}^{n} \left(\frac{\left\|p^{(k)} - q\right\|_2}{\left\|p^{(i)} - q\right\|_2}\right)^m$$

$$< \sum_{i'=1}^{\lceil \log_2(n/k) \rceil - 1} 2^{i'} k \gamma^{-i'm}$$

If $\gamma \geq \sqrt[m]{2}$, this quantity is at most $k \log_2(n/k)$. On the other hand, if $1 \leq \gamma < \sqrt[m]{2}$, this quantity can be simplified to:

$$
\begin{aligned}
& k\left(\frac{2}{\gamma^m}\right)\left(\left(\frac{2}{\gamma^m}\right)^{\lceil \log_2(n/k)\rceil - 1} - 1\right) / \left(\frac{2}{\gamma^m} - 1\right) \\
&= O\left(k\left(\frac{2}{\gamma^m}\right)^{\lceil \log_2(n/k)\rceil - 1}\right) \\
&= O\left(k\left(\frac{n}{k}\right)^{1 - m\log_2\gamma}\right)
\end{aligned}
$$

Therefore, $\sum_{i=2k+1}^{n} \left(\left\|p^{(k)} - q\right\|_2 / \left\|p^{(i)} - q\right\|_2\right)^m \leq O\left(k \max(\log(n/k), (n/k)^{1-m\log_2\gamma})\right)$. □

Lemma 9. *For a dataset with global relative sparsity (k, γ) and a given composite index consisting of m simple indices, there is some $k_0 \in \Omega(k \max(\log(n/k), (n/k)^{1-m\log_2\gamma}))$ such that the probability that the candidate points retrieved from the composite index do not include some of the true k-nearest neighbours is at most some constant $\alpha_0 < 1$.*

Proof. We will refer to the true k-nearest neighbours that are among first k_0 points retrieved from the composite index as *true positives* and those that are not as *false negatives*. Additionally, we will refer to points that are not true k-nearest neighbours but are among the first k_0 points retrieved as *false positives*.

When not all the true k-nearest neighbours are among the first k_0 candidate points, there must be at least one false negative and so there can be at most $k-1$ true positives. Consequently, there must be at least $k_0 - (k-1)$ false positives. To find an upper bound on the probability of the existence of $k_0-(k-1)$ false positives in terms of global relative sparsity, we apply Lemma 6 with n_0 set to $k_0 - (k-1)$, followed by Lemma 8. We conclude that this probability is at most $\frac{1}{k_0-2k+1} O\left(k \max(\log(n/k), (n/k)^{1-m\log_2\gamma})\right)$. Because the event that not all the true k-nearest neighbours are among the first k_0 candidate points is contained in the event that there are $k_0 - (k-1)$ false positives, the former is upper bounded by the same quantity. So, we can choose some $k_0 \in \Omega(k \max(\log(n/k), (n/k)^{1-m\log_2\gamma}))$ to make it strictly less than 1. □

Lemma 10. *For a dataset with global relative sparsity (k, γ) and a given composite index consisting of m simple indices, there is some $k_1 \in \Omega(mk \max(\log(n/k), (n/k)^{1-\log_2\gamma}))$ such that the probability that the candidate points retrieved from the composite index do not include some of the true k-nearest neighbours is at most some constant $\alpha_1 < 1$.*

Proof. We will refer to the projections of true k-nearest neighbours that are among first k_1 visited point projections as *true positives* and those that are not as *false negatives*. Additionally, we will refer to projections of points that are not of the true k-nearest neighbours but are among the first k_1 visited point projections as *false positives*.

When a k-nearest neighbour is not among the candidate points that have been retrieved, some of its projections must not be among the first k_1 visited point projections. So, there must be at least one false negative, implying that there can be at most $mk-1$ true positives. Consequently, there must be at least $k_1 - (mk-1)$ false positives. To find an upper bound on the probability of the existence of $k_1 - (mk-1)$ false positives in terms of global relative

sparsity, we apply Lemma 7 with n_0 set to $k_1-(mk-1)$, followed by Lemma 8. We conclude that this probability is at most $\frac{m}{k_1-2mk+1}O\left(k\max(\log(n/k),(n/k)^{1-\log_2\gamma})\right)$. Because the event that some true k-nearest neighbour is missing from the candidate points is contained in the event that there are $k_1-(mk-1)$ false positives, the former is upper bounded by the same quantity. So, we can choose some $k_1 \in \Omega(mk\max(\log(n/k),(n/k)^{1-\log_2\gamma}))$ to make it strictly less than 1. □

Theorem 10. *For a dataset with global relative sparsity (k,γ), for any $\epsilon>0$, there is some L, $k_0 \in \Omega(k\max(\log(n/k),(n/k)^{1-m\log_2\gamma}))$ and $k_1 \in \Omega(mk\max(\log(n/k),(n/k)^{1-\log_2\gamma}))$ such that the algorithm returns the correct set of k-nearest neighbours with probability of at least $1-\epsilon$.*

Proof. For a given composite index, by Lemma 9, there is some $k_0 \in \Omega(k\max(\log(n/k),(n/k)^{1-m\log_2\gamma}))$ such that the probability that some of the true k-nearest neighbours are missed is at most some constant $\alpha_0<1$. Likewise, by Lemma 10, there is some $k_1 \in \Omega(mk\max(\log(n/k),(n/k)^{1-\log_2\gamma}))$ such that this probability is at most some constant $\alpha_1<1$. By choosing such k_0 and k_1, this probability is therefore at most $\min\{\alpha_0,\alpha_1\}<1$. For the algorithm to fail, all composite indices must miss some k-nearest neighbours. Since each composite index is constructed independently, the algorithm fails with probability of at most $(\min\{\alpha_0,\alpha_1\})^L$, and so must succeed with probability of at least $1-(\min\{\alpha_0,\alpha_1\})^L$. Since $\min\{\alpha_0,\alpha_1\}<1$, there is some L that makes $1-(\min\{\alpha_0,\alpha_1\})^L \geq 1-\epsilon$. □

Theorem 11. *For a given number of simple indices m, the algorithm takes $O\left(d(m+k\max(\log(n/k),(n/k)^{1-m/d'}))+mk\log m\left(\max(\log(n/k),(n/k)^{1-1/d'})\right)\right)$ time to retrieve the k-nearest neighbours at query time, where d' denotes the intrinsic dimensionality.*

Proof. Computing projections of the query point along all u_{jl}'s takes $O(dm)$ time, since L is a constant. Searching in the binary search trees/skip lists T_{jl}'s takes $O(m\log n)$ time. The total number of point projections that are visited is at most $\Theta(mk\max(\log(n/k),(n/k)^{1-\log_2\gamma}))$. Because determining the next point to visit requires popping and pushing a priority queue, which takes $O(\log m)$ time, the total time spent on visiting points is $O(mk\log m\max(\log(n/k),(n/k)^{1-\log_2\gamma}))$. The total number of candidate points retrieved is at most $\Theta(k\max(\log(n/k),(n/k)^{1-m\log_2\gamma}))$. Because true distances are computed for every candidate point, the total time spent on distance computation is $O(dk\max(\log(n/k),(n/k)^{1-m\log_2\gamma}))$. We can find the k closest points to the query among the candidate points using a selection algorithm like quickselect, which takes $O(k\max(\log(n/k),(n/k)^{1-m\log_2\gamma}))$ time on average. So, the entire algorithm takes $O(d(m+k\max(\log(n/k),(n/k)^{1-m\log_2\gamma}))+mk\log m\max(\log(n/k),(n/k)^{1-\log_2\gamma}))$ time. Substituting $1/d'$ for $\log_2\gamma$ yields the desired expression. □

Theorem 12. *For a given number of simple indices m, the algorithm takes $O(m(dn+n\log n))$ time to preprocess the data points in D at construction time.*

Proof. Computing projections of all n points along all u_{jl}'s takes $O(dmn)$ time, since L is a constant. Inserting all n points into mL self-balancing binary search trees/skip lists takes $O(mn \log n)$ time. □

Theorem 13. *The algorithm requires $O(m(d + \log n))$ time to insert a new data point and $O(m \log n)$ time to delete a data point.*

Proof. In order to insert a data point, we need to compute its projection along all u_{jl}'s and insert it into each binary search tree or skip list. Computing the projections takes $O(md)$ time and inserting them into the corresponding self-balancing binary search trees or skip lists takes $O(m \log n)$ time. In order to delete a data point, we simply remove its projections from each of the binary search trees or skip lists, which takes $O(m \log n)$ time. □

Theorem 14. *The algorithm requires $O(mn)$ space in addition to the space used to store the data.*

Proof. The only additional information that needs to be stored are the mL binary search trees or skip lists. Since n entries are stored in each binary search tree/skip list, the total additional space required is $O(mn)$. □

Appendix B

Implicit Maximum Likelihood Estimation

This section contains the proof of the result presented in the Implicit Maximum Likelihood Estimation section of the dissertation. Before proving the main result, we first prove the following intermediate results:

Lemma 13. *Let $\Omega \subseteq \mathbb{R}^d$ and $V \subseteq \mathbb{R}$. For $i \in [N]$, let $f_i : \Omega \to V$ be differentiable on Ω and $\Phi : V \to \mathbb{R}$ be differentiable on V and strictly increasing. Assume $\arg\min_{\theta\in\Omega} \sum_{i=1}^N f_i(\theta)$ exists and is unique. Let $\theta^* \coloneqq \arg\min_{\theta\in\Omega} \sum_{i=1}^N f_i(\theta)$ and $w_i \coloneqq 1/\Phi'(f_i(\theta^*))$. If the following conditions hold:*

- *There is a bounded set $S \subseteq \Omega$ such that $\mathrm{bd}(S) \subseteq \Omega$, $\theta^* \in S$ and $\forall f_i,\ \forall \theta \in \Omega \backslash S,\ f_i(\theta) > f_i(\theta^*)$, where $\mathrm{bd}(S)$ denotes the boundary of S.*
- *For all $\theta \in \Omega$, if $\theta \neq \theta^*$, there exists $j \in [d]$ such that*
$$\left\langle \begin{pmatrix} w_1\Phi'(f_1(\theta)) \\ \vdots \\ w_n\Phi'(f_n(\theta)) \end{pmatrix}, \begin{pmatrix} \partial f_1/\partial\theta_j(\theta) \\ \vdots \\ \partial f_n/\partial\theta_j(\theta) \end{pmatrix} \right\rangle \neq 0.$$

Then $\arg\min_{\mathbf{x}\in\boldsymbol{\Omega}} \sum_{i=1}^N w_i\Phi(f_i(\theta))$ exists and is unique. Furthermore, $\arg\min_{\theta\in\Omega} \sum_{i=1}^N w_i\Phi(f_i(\theta)) = \arg\min_{\theta\in\Omega} \sum_{i=1}^N f_i(\theta)$.

Proof. Let $S \subseteq \Omega$ be the bounded set such that $\mathrm{bd}(S) \subseteq \Omega$, $\theta^* \in S$ and $\forall f_i,\ \forall \theta \in \Omega \setminus S,\ f_i(\theta) > f_i(\theta^*)$. Consider the closure of $S \coloneqq S \cup \mathrm{bd}(S)$, denoted as $\bar{S}$. Because $S \subseteq \Omega$ and $\mathrm{bd}(S) \subseteq \Omega$, $\bar{S} \subseteq \Omega$. Since S is bounded, $\bar{S}$ is bounded. Because $\bar{S} \subseteq \Omega \subseteq \mathbb{R}^d$ and is closed and bounded, it is compact.

Consider the function $\sum_{i=1}^N w_i\Phi(f_i(\cdot))$. By the differentiability of f_i's and Φ, $\sum_{i=1}^N w_i\Phi(f_i(\cdot))$ is differentiable on Ω and hence continuous on Ω. By the compactness of $\bar{S}$ and the continuity of $\sum_{i=1}^N w_i\Phi(f_i(\cdot))$ on $\bar{S} \subseteq \Omega$, Extreme Value Theorem applies, which implies that

$\min_{\theta\in\bar{S}} \sum_{i=1}^{N} w_i\Phi(f_i(\theta))$ exists. Let $\tilde{\theta} \in \bar{S}$ be such that $\sum_{i=1}^{N} w_i\Phi(f_i(\tilde{\theta})) = \min_{\theta\in\bar{S}} \sum_{i=1}^{N} w_i\Phi(f_i(\theta))$.

By definition of S, $\forall f_i,\ \forall\theta \in \Omega \setminus S,\ f_i(\theta) > f_i(\theta^*)$, implying that $\Phi(f_i(\theta)) > \Phi(f_i(\theta^*))$ since Φ is strictly increasing. Because $\Phi'(\cdot) > 0$, $w_i > 0$ and so $\sum_{i=1}^{N} w_i\Phi(f_i(\theta)) > \sum_{i=1}^{N} w_i\Phi(f_i(\theta^*))\ \forall\theta \in \Omega \setminus S$. At the same time, since $\theta^* \in S \subset \bar{S}$, by definition of $\tilde{\theta}$, $\sum_{i=1}^{N} w_i\Phi(f_i(\tilde{\theta})) \leq \sum_{i=1}^{N} w_i\Phi(f_i(\theta^*))$. Combining these two facts yields $\sum_{i=1}^{N} w_i\Phi(f_i(\tilde{\theta})) \leq \sum_{i=1}^{N} w_i\Phi(f_i(\theta^*)) < \sum_{i=1}^{N} w_i\Phi(f_i(\theta))\ \forall\theta \in \Omega \setminus S$. Since the inequality is strict, this implies that $\tilde{\theta} \notin \Omega \setminus S$, and so $\tilde{\theta} \in \bar{S} \setminus (\Omega \setminus S) \subseteq \Omega \setminus (\Omega \setminus S) = S$.

In addition, because $\tilde{\theta}$ is the minimizer of $\sum_{i=1}^{N} w_i\Phi(f_i(\cdot))$ on $\bar{S}$, $\sum_{i=1}^{N} w_i\Phi(f_i(\tilde{\theta})) \leq \sum_{i=1}^{N} w_i\Phi(f_i(\theta))\ \forall\theta \in \bar{S}$. So, $\sum_{i=1}^{N} w_i\Phi(f_i(\tilde{\theta})) \leq \sum_{i=1}^{N} w_i\Phi(f_i(\theta))\ \forall\theta \in \bar{S} \cup (\Omega \setminus S) \supseteq S \cup (\Omega \setminus S) = \Omega$. Hence, $\tilde{\theta}$ is a minimizer of $\sum_{i=1}^{N} w_i\Phi(f_i(\cdot))$ on Ω, and so $\min_{\theta\in\Omega} \sum_{i=1}^{N} w_i\Phi(f_i(\theta))$ exists. Because $\sum_{i=1}^{N} w_i\Phi(f_i(\cdot))$ is differentiable on Ω, $\tilde{\theta}$ must be a critical point of $\sum_{i=1}^{N} w_i\Phi(f_i(\cdot))$ on Ω.

On the other hand, since Φ is differentiable on V and $f_i(\theta) \in V$ for all $\theta \in \Omega$, $\Phi'(f_i(\theta))$ exists for all $\theta \in \Omega$. So,

$$\begin{aligned}\nabla\left(\sum_{i=1}^{N} w_i\Phi(f_i(\theta))\right) &= \sum_{i=1}^{N} w_i\nabla\left(\Phi(f_i(\theta))\right)\\ &= \sum_{i=1}^{N} w_i\Phi'(f_i(\theta))\nabla f_i(\theta)\\ &= \sum_{i=1}^{N} \frac{\Phi'(f_i(\theta))}{\Phi'(f_i(\theta^*))}\nabla f_i(\theta)\end{aligned}$$

At $\theta = \theta^*$,

$$\begin{aligned}\nabla\left(\sum_{i=1}^{N} w_i\Phi(f_i(\theta^*))\right) &= \sum_{i=1}^{N} \frac{\Phi'(f_i(\theta^*))}{\Phi'(f_i(\theta^*))}\nabla f_i(\theta^*)\\ &= \sum_{i=1}^{N} \nabla f_i(\theta^*)\end{aligned}$$

Since each f_i is differentiable on Ω, $\sum_{i=1}^{N} f_i$ is differentiable on Ω. Combining this with the fact that θ^* is the minimizer of $\sum_{i=1}^{N} f_i$ on Ω, it follows that $\nabla\left(\sum_{i=1}^{N} f_i(\theta^*)\right) = \sum_{i=1}^{N} \nabla f_i(\theta^*) = 0$. Hence, $\nabla\left(\sum_{i=1}^{N} w_i\Phi(f_i(\theta^*))\right) = 0$ and so θ^* is a critical point of $\sum_{i=1}^{N} w_i\Phi(f_i(\cdot))$.

Because $\forall\theta \in \Omega$, if $\theta \neq \theta^*$, $\exists j \in [d]$ such that $\left\langle \begin{pmatrix} w_1\Phi'(f_1(\theta)) \\ \vdots \\ w_n\Phi'(f_n(\theta)) \end{pmatrix}, \begin{pmatrix} \partial f_1/\partial\theta_j(\theta) \\ \vdots \\ \partial f_n/\partial\theta_j(\theta) \end{pmatrix} \right\rangle \neq 0$, $\sum_{i=1}^{N} w_i\Phi'(f_i(\theta))\nabla f_i(\theta) = \nabla\left(\sum_{i=1}^{N} w_i\Phi(f_i(\theta))\right) \neq 0$ for any $\theta \neq \theta^* \in \Omega$. Therefore, θ^*

is the only critical point of $\sum_{i=1}^{N} w_i \Phi(f_i(\cdot))$ on Ω. Since $\tilde{\theta}$ is a critical point on Ω, we can conclude that $\theta^* = \tilde{\theta}$, and so θ^* is a minimizer of $\sum_{i=1}^{N} w_i \Phi(f_i(\cdot))$ on Ω. Since any other minimizer must be a critical point and θ^* is the only critical point, θ^* is the unique minimizer. So, $\arg\min_{\theta\in\Omega} \sum_{i=1}^{N} f_i(\theta) = \theta^* = \arg\min_{\theta\in\Omega} \sum_{i=1}^{N} w_i \Phi(f_i(\theta))$. □

Lemma 14. *Let P be a distribution on $\mathbb{R}^d$ whose density $p(\cdot)$ is continuous at a point $\mathbf{x}_0 \in \mathbb{R}^d$ and $\mathbf{x} \sim P$ be a random variable. Let $\tilde{r} \coloneqq \|\mathbf{x} - \mathbf{x}_0\|_2$, $\kappa \coloneqq \pi^{d/2}/\Gamma\left(\frac{d}{2}+1\right)$, where $\Gamma(\cdot)$ denotes the gamma function*[1]*, and $r \coloneqq \kappa \tilde{r}^d$. Let $G(\cdot)$ denote the cumulative distribution function (CDF) of r and $\partial_+ G(\cdot)$ denote the one-sided derivative of G from the right. Then, $\partial_+ G(0) = p(\mathbf{x}_0)$.*

Proof. By definition of $\partial_+ G(\cdot)$,

$$\begin{aligned}\partial_+ G(0) &= \lim_{h\to 0^+} \frac{G(h) - G(0)}{h} = \lim_{h\to 0^+} \frac{G(h)}{h} \\ &= \lim_{h\to 0^+} \frac{\Pr\left(r \le h\right)}{h} = \lim_{h\to 0^+} \frac{\Pr\left(\tilde{r} \le \sqrt[d]{h/\kappa}\right)}{h}\end{aligned}$$

If we define $\tilde{h} \coloneqq \sqrt[d]{h/\kappa}$, the above can be re-written as:

$$\partial_+ G(0) = \lim_{\tilde{h}\to 0^+} \frac{\Pr\left(\tilde{r} \le \tilde{h}\right)}{\kappa \tilde{h}^d} = \lim_{\tilde{h}\to 0^+} \frac{\int_{B_{\mathbf{x}_0}(\tilde{h})} p(\mathbf{u}) d\mathbf{u}}{\kappa \tilde{h}^d}$$

We want to show that $\lim_{\tilde{h}\to 0^+} \left(\int_{B_{\mathbf{x}_0}(\tilde{h})} p(\mathbf{u}) d\mathbf{u}\right) / \kappa \tilde{h}^d = p(\mathbf{x}_0)$. In other words, we want to show $\forall \epsilon > 0\ \exists \delta > 0$ such that $\forall \tilde{h} \in (0, \delta)$, $\left| \frac{\int_{B_{\mathbf{x}_0}(\tilde{h})} p(\mathbf{u}) d\mathbf{u}}{\kappa \tilde{h}^d} - p(\mathbf{x}_0) \right| < \epsilon$.

Let $\epsilon > 0$ be arbitrary.

Since $p(\cdot)$ is continuous at $\mathbf{x}_0$, by definition, $\forall \tilde{\epsilon} > 0\ \exists \tilde{\delta} > 0$ such that $\forall \mathbf{u} \in B_{\mathbf{x}_0}(\tilde{\delta})$, $|p(\mathbf{u}) - p(\mathbf{x}_0)| < \tilde{\epsilon}$. Let $\tilde{\delta} > 0$ be such that $\forall \mathbf{u} \in B_{\mathbf{x}_0}(\tilde{\delta})$, $p(\mathbf{x}_0) - \epsilon < p(\mathbf{u}) < p(\mathbf{x}_0) + \epsilon$. We choose $\delta = \tilde{\delta}$.

Let $0 < \tilde{h} < \delta$ be arbitrary. Since $p(\mathbf{x}_0) - \epsilon < p(\mathbf{u}) < p(\mathbf{x}_0) + \epsilon\ \forall \mathbf{u} \in B_{\mathbf{x}_0}(\tilde{\delta}) = B_{\mathbf{x}_0}(\delta) \supset B_{\mathbf{x}_0}(\tilde{h})$,

$$\begin{aligned}\int_{B_{\mathbf{x}_0}(\tilde{h})} p(\mathbf{u}) d\mathbf{u} &< \int_{B_{\mathbf{x}_0}(\tilde{h})} \left(p(\mathbf{x}_0) + \epsilon\right) d\mathbf{u} \\ &= \left(p(\mathbf{x}_0) + \epsilon\right) \int_{B_{\mathbf{x}_0}(\tilde{h})} d\mathbf{u}\end{aligned}$$

[1] The constant κ is the the ratio of the volume of a d-dimensional ball of radius $\tilde{r}$ to a d-dimensional cube of side length $\tilde{r}$.

Observe that $\int_{B_{\mathbf{x}_0}(\tilde{h})} d\mathbf{u}$ is the volume of a d-dimensional ball of radius $\tilde{h}$, so $\int_{B_{\mathbf{x}_0}(\tilde{h})} d\mathbf{u} = \kappa\tilde{h}^d$. Thus, $\int_{B_{\mathbf{x}_0}(\tilde{h})} p(\mathbf{u})d\mathbf{u} < \kappa\tilde{h}^d\left(p(\mathbf{x}_0)+\epsilon\right)$, implying that $\left(\int_{B_{\mathbf{x}_0}(\tilde{h})} p(\mathbf{u})d\mathbf{u}\right)/\kappa\tilde{h}^d < p(\mathbf{x}_0)+\epsilon$. By similar reasoning, we conclude that $\left(\int_{B_{\mathbf{x}_0}(\tilde{h})} p(\mathbf{u})d\mathbf{u}\right)/\kappa\tilde{h}^d > p(\mathbf{x}_0)-\epsilon$.

Hence,

$$\left|\frac{\int_{B_{\mathbf{x}_0}(\tilde{h})} p(\mathbf{u})d\mathbf{u}}{\kappa\tilde{h}^d} - p(\mathbf{x}_0)\right| < \epsilon \;\forall \tilde{h} \in (0,\delta)$$

Therefore,

$$\partial_+ G(0) = \lim_{\tilde{h}\to 0^+} \frac{\int_{B_{\mathbf{x}_0}(\tilde{h})} p(\mathbf{u})d\mathbf{u}}{\kappa\tilde{h}^d} = p(\mathbf{x}_0)$$

□

Lemma 15. *Let P_θ be a parameterized family of distributions on $\mathbb{R}^d$ with parameter θ and probability density function (PDF) $p_\theta(\cdot)$ that is continuous at a point $\mathbf{x}_i$. Consider a random variable $\tilde{\mathbf{x}}_1^\theta \sim P_\theta$ and define $\tilde{r}_i^\theta := \left\|\tilde{\mathbf{x}}_1^\theta - \mathbf{x}_i\right\|_2^2$, whose cumulative distribution function (CDF) is denoted by $F_i^\theta(\cdot)$. Assume P_θ has the following property: for any θ_1, θ_2, there exists θ_0 such that $F_i^{\theta_0}(t) \geq \max\left\{F_i^{\theta_1}(t), F_i^{\theta_2}(t)\right\}\ \forall t \geq 0$ and $p_{\theta_0}(\mathbf{x}_i) = \max\left\{p_{\theta_1}(\mathbf{x}_i), p_{\theta_2}(\mathbf{x}_i)\right\}$. For any $m \geq 1$, let $\tilde{\mathbf{x}}_1^\theta, \ldots, \tilde{\mathbf{x}}_m^\theta \sim P_\theta$ be i.i.d. random variables and define $R_i^\theta := \min_{j\in[m]} \left\|\tilde{\mathbf{x}}_j^\theta - \mathbf{x}_i\right\|_2^2$. Then the function $\Psi_i : z \mapsto \min_\theta\left\{\mathbb{E}\left[R_i^\theta\right] | p_\theta(\mathbf{x}_i) = z\right\}$ is strictly decreasing.*

Proof. Let $r_i^\theta := \kappa\left(\tilde{r}_i^\theta\right)^{d/2} = \kappa\left\|\tilde{\mathbf{x}}_1^\theta - \mathbf{x}_i\right\|_2^d$ be a random variable and let $G_i^\theta(\cdot)$ be the CDF of r_i^θ. Since R_i^θ is nonnegative,

$$\begin{aligned}
\mathbb{E}\left[R_i^\theta\right] &= \int_0^\infty \Pr\left(R_i^\theta > t\right) dt \\
&= \int_0^\infty \left(\Pr\left(\left\|\tilde{\mathbf{x}}_1^\theta - \mathbf{x}_i\right\|_2^2 > t\right)\right)^m dt \\
&= \int_0^\infty \left(\Pr\left(\kappa\left\|\tilde{\mathbf{x}}_1^\theta - \mathbf{x}_i\right\|_2^d > \kappa t^{d/2}\right)\right)^m dt \\
&= \int_0^\infty \left(\Pr\left(r_i^\theta > \kappa t^{d/2}\right)\right)^m dt \\
&= \int_0^\infty \left(1 - G_i^\theta\left(\kappa t^{d/2}\right)\right)^m dt
\end{aligned}$$

Also, by Lemma 14, $p_\theta(\mathbf{x}_i) = \partial_+ G_i^\theta(0)$. Using these facts, we can rewrite $\min_\theta\left\{\mathbb{E}\left[R_i^\theta\right] | p_\theta(\mathbf{x}_i) = z\right\}$ as $\min_\theta\left\{\int_0^\infty \left(1 - G_i^\theta\left(\kappa t^{d/2}\right)\right)^m dt \,\middle|\, \partial_+ G_i^\theta(0) = z\right\}$. By definition of Ψ_i, $\min_\theta\left\{\int_0^\infty \left(1 - G_i^\theta\left(\kappa t^{d/2}\right)\right)^m dt \,\middle|\, \partial_+ G_i^\theta(0) = z\right\}$ exists for all z. Let $\phi_i(z)$ be a value of θ that attains the minimum. Define $G_i^*(y,z) := G_i^{\phi_i(z)}(y)$. By definition, $\frac{\partial_+}{\partial y} G_i^*(0,z) = z$, where

$\frac{\partial_+}{\partial y}G_i^*(y,z)$ denotes the one-sided partial derivative from the right w.r.t. y. Also, since $G_i^*(\cdot,z)$ is the CDF of a distribution of a non-negative random variable, $G_i^*(0,z)=0$.

By definition of $\frac{\partial_+}{\partial y}G_i^*(0,z)$, $\forall\epsilon>0\ \exists\delta>0$ such that $\forall h\in(0,\delta)$, $\left|\frac{G_i^*(h,z)-G_i^*(0,z)}{h}-z\right|<\epsilon$.

Let $z'>z$. Let $\delta>0$ be such that $\forall h\in(0,\delta)$, $\left|\frac{G_i^*(h,z)-G_i^*(0,z)}{h}-z\right|<\frac{z'-z}{2}$ and $\delta'>0$ be such that $\forall h\in(0,\delta')$, $\left|\frac{G_i^*(h,z')-G_i^*(0,z')}{h}-z'\right|<\frac{z'-z}{2}$.

Consider $h\in(0,\min(\delta,\delta'))$. Then, $\frac{G_i^*(h,z)-G_i^*(0,z)}{h}=\frac{G_i^*(h,z)}{h}<z+\frac{z'-z}{2}=\frac{z+z'}{2}$ and $\frac{G_i^*(h,z')-G_i^*(0,z')}{h}=\frac{G_i^*(h,z')}{h}>z'-\frac{z'-z}{2}=\frac{z+z'}{2}$. So,

$$\frac{G_i^*(h,z)}{h}<\frac{z+z'}{2}<\frac{G_i^*(h,z')}{h}$$

Multiplying by h on both sides, we conclude that $G_i^*(h,z)<G_i^*(h,z')\ \forall h\in(0,\min(\delta,\delta'))$.

Let $\alpha\coloneqq\sqrt[d]{\min(\delta,\delta')/\kappa}$. We can break $\int_0^\infty\left(1-G_i^*\left(\kappa t^{d/2},z\right)\right)^m dt$ into two terms:

$$\begin{aligned}&\int_0^\infty\left(1-G_i^*\left(\kappa t^{d/2},z\right)\right)^m dt\\=&\int_0^\alpha\left(1-G_i^*\left(\kappa t^{d/2},z\right)\right)^m dt+\int_\alpha^\infty\left(1-G_i^*\left(\kappa t^{d/2},z\right)\right)^m dt\end{aligned}$$

We can also do the same for $\int_0^\infty\left(1-G_i^*\left(\kappa t^{d/2},z'\right)\right)^m dt$.

Because $G_i^*(h,z)<G_i^*(h,z')\ \forall h\in(0,\min(\delta,\delta'))$, $G_i^*(\kappa t^{d/2},z)<G_i^*(\kappa t^{d/2},z')\ \forall t\in(0,\alpha)$. It follows that $1-G_i^*(\kappa t^{d/2},z)>1-G_i^*(\kappa t^{d/2},z')$ and $\left(1-G_i^*\left(\kappa t^{d/2},z\right)\right)^m>\left(1-G_i^*(\kappa t^{d/2},z')\right)^m\ \forall t\in(0,\alpha)$. So, $\int_0^\alpha\left(1-G_i^*\left(\kappa t^{d/2},z\right)\right)^m dt>\int_0^\alpha\left(1-G_i^*\left(\kappa t^{d/2},z'\right)\right)^m dt$.

We now consider the second term. First, observe that $F_i^\theta(t)=\Pr\left(\left\|\tilde{\mathbf{x}}_1^\theta-\mathbf{x}_i\right\|_2^2\le t\right)=\Pr\left(\kappa\left\|\tilde{\mathbf{x}}_1^\theta-\mathbf{x}_i\right\|_2^d\le\kappa t^{d/2}\right)=G_i^\theta\left(\kappa t^{d/2}\right)$ for all $t\ge0$. So, by the property of P_θ, for any θ_1,θ_2, there exists θ_0 such that $G_i^{\theta_0}(\kappa t^{d/2})=F_i^{\theta_0}(t)\ge\max\left\{F_i^{\theta_1}(t),F_i^{\theta_2}(t)\right\}$ $=\max\left\{G_i^{\theta_1}(\kappa t^{d/2}),G_i^{\theta_2}(\kappa t^{d/2})\right\}\ \forall t\ge0$ and $\partial_+G_i^{\theta_0}(0)=p_{\theta_0}(\mathbf{x}_i)=\max\left\{p_{\theta_1}(\mathbf{x}_i),p_{\theta_2}(\mathbf{x}_i)\right\}=\max\left\{\partial_+G_i^{\theta_1}(0),\partial_+G_i^{\theta_2}(0)\right\}$.

Take $\theta_1=\phi_i(z)$ and $\theta_2=\phi_i(z')$. Let θ_0 be such that $G_i^{\theta_0}(\kappa t^{d/2})\ge$ $\max\left\{G_i^{\theta_1}(\kappa t^{d/2}),G_i^{\theta_2}(\kappa t^{d/2})\right\}\ \forall t\ge0$ and $\partial_+G_i^{\theta_0}(0)=\max\left\{\partial_+G_i^{\theta_1}(0),\partial_+G_i^{\theta_2}(0)\right\}$. By definition of $\phi_i(\cdot)$, $\partial_+G_i^{\theta_1}(0)=z$ and $\partial_+G_i^{\theta_2}(0)=z'$. So, $\partial_+G_i^{\theta_0}(0)=\max\{z,z'\}=z'$. Since $G_i^{\theta_0}(\kappa t^{d/2})\ge G_i^{\theta_2}(\kappa t^{d/2})\ \forall t\ge0$, $1-G_i^{\theta_0}\left(\kappa t^{d/2}\right)\le1-G_i^{\theta_2}\left(\kappa t^{d/2}\right)\ \forall t\ge0$ and so $\int_0^\infty\left(1-G_i^{\theta_0}\left(\kappa t^{d/2}\right)\right)^m dt\le\int_0^\infty\left(1-G_i^{\theta_2}\left(\kappa t^{d/2}\right)\right)^m dt$. On the other hand, because $\theta_2=\phi_i(z')$ minimizes $\int_0^\infty\left(1-G_i^\theta\left(\kappa t^{d/2}\right)\right)^m dt$ among all θ's such that $\partial_+G_i^\theta(0)=z'$ and $\partial_+G_i^{\theta_0}(0)=z'$, $\int_0^\infty\left(1-G_i^{\theta_2}\left(\kappa t^{d/2}\right)\right)^m dt\le\int_0^\infty\left(1-G_i^{\theta_0}\left(\kappa t^{d/2}\right)\right)^m dt$. We can therefore conclude that $\int_0^\infty\left(1-G_i^{\theta_0}\left(\kappa t^{d/2}\right)\right)^m dt=\int_0^\infty\left(1-G_i^{\theta_2}\left(\kappa t^{d/2}\right)\right)^m dt$. Since $1-G_i^{\theta_0}\left(\kappa t^{d/2}\right)\le1-G_i^{\theta_2}\left(\kappa t^{d/2}\right)$ $\forall t\ge0$, the only situation where this can happen is when $G_i^{\theta_0}\left(\kappa t^{d/2}\right)=G_i^{\theta_2}\left(\kappa t^{d/2}\right)\forall t\ge0$.

By definition of G_i^*, $G_i^*\left(\kappa t^{d/2},z\right)=G_i^{\phi_i(z)}(\kappa t^{d/2})=G_i^{\theta_1}(\kappa t^{d/2})$ and $G_i^*\left(\kappa t^{d/2},z'\right)=G_i^{\phi_i(z')}(\kappa t^{d/2})=G_i^{\theta_2}(\kappa t^{d/2})=G_i^{\theta_0}\left(\kappa t^{d/2}\right)$. By definition of θ_0, $G_i^{\theta_0}\left(\kappa t^{d/2}\right)\ge G_i^{\theta_1}(\kappa t^{d/2})\ \forall t\ge$

0. So, $G_i^*\left(\kappa t^{d/2}, z'\right) = G_i^{\theta_2}(\kappa t^{d/2}) \geq G_i^{\theta_1}(\kappa t^{d/2}) = G_i^*\left(\kappa t^{d/2}, z\right)\ \forall t \geq 0$. Hence, $\int_\alpha^\infty \left(1 - G_i^*\left(\kappa t^{d/2}, z'\right)\right)^m dt \leq \int_\alpha^\infty \left(1 - G_i^*\left(\kappa t^{d/2}, z\right)\right)^m dt$.

Combining with the previous result that $\int_0^\alpha \left(1 - G_i^*\left(\kappa t^{d/2}, z'\right)\right)^m dt$ $< \int_0^\alpha \left(1 - G_i^*\left(\kappa t^{d/2}, z\right)\right)^m dt$, it follows that:

$$\begin{aligned}
&\int_0^\infty \left(1 - G_i^*\left(\kappa t^{d/2}, z'\right)\right)^m dt \\
&= \int_0^\alpha \left(1 - G_i^*\left(\kappa t^{d/2}, z'\right)\right)^m dt + \int_\alpha^\infty \left(1 - G_i^*\left(\kappa t^{d/2}, z'\right)\right)^m dt \\
&< \int_0^\alpha \left(1 - G_i^*\left(\kappa t^{d/2}, z\right)\right)^m dt + \int_\alpha^\infty \left(1 - G_i^*\left(\kappa t^{d/2}, z'\right)\right)^m dt \\
&\leq \int_0^\alpha \left(1 - G_i^*\left(\kappa t^{d/2}, z\right)\right)^m dt + \int_\alpha^\infty \left(1 - G_i^*\left(\kappa t^{d/2}, z\right)\right)^m dt \\
&= \int_0^\infty \left(1 - G_i^*\left(\kappa t^{d/2}, z\right)\right)^m dt
\end{aligned}$$

By definition,

$$\begin{aligned}
&\int_0^\infty \left(1 - G_i^*\left(\kappa t^{d/2}, z\right)\right)^m dt \\
&= \int_0^\infty \left(1 - G_i^{\phi_i(z)}(\kappa t^{d/2})\right)^m dt \\
&= \min_\theta \left\{ \int_0^\infty \left(1 - G_i^\theta\left(\kappa t^{d/2}\right)\right)^m dt \,\middle|\, \partial_+ G_i^\theta(0) = z \right\} \\
&= \min_\theta \left\{ \mathbb{E}\left[R_i^\theta\right] | p_\theta(\mathbf{x}_i) = z \right\} \\
&= \Psi_i(z)
\end{aligned}$$

Similarly, $\int_0^\infty \left(1 - G_i^*\left(\kappa t^{d/2}, z'\right)\right)^m dt = \Psi_i(z')$. We can therefore conclude that $\Psi_i(z') < \Psi_i(z)$ whenever $z' > z$. □

We now prove the main result.

Theorem 1. *Consider a set of observations $\mathbf{x}_1, \ldots, \mathbf{x}_n$, a parameterized family of distributions P_θ with probability density function (PDF) $p_\theta(\cdot)$ and a unique maximum likelihood solution θ^*. For any $m \geq 1$, let $\tilde{\mathbf{x}}_1^\theta, \ldots, \tilde{\mathbf{x}}_m^\theta \sim P_\theta$ be i.i.d. random variables and define $\tilde{r}^\theta \coloneqq \left\|\tilde{\mathbf{x}}_1^\theta\right\|_2^2$, $R^\theta \coloneqq \min_{j\in[m]} \left\|\tilde{\mathbf{x}}_j^\theta\right\|_2^2$ and $R_i^\theta \coloneqq \min_{j\in[m]} \left\|\tilde{\mathbf{x}}_j^\theta - \mathbf{x}_i\right\|_2^2$. Let $F^\theta(\cdot)$ be the cumulative distribution function (CDF) of $\tilde{r}^\theta$ and $\Psi(z) \coloneqq \min_\theta \left\{\mathbb{E}\left[R^\theta\right] | p_\theta(\mathbf{0}) = z\right\}$.*

If P_θ satisfies the following:

- *$p_\theta(\mathbf{x})$ is differentiable w.r.t. θ and continuous w.r.t. $\mathbf{x}$ everywhere.*
- *$\forall \theta, \mathbf{v}$, there exists θ' such that $p_\theta(\mathbf{x}) = p_{\theta'}(\mathbf{x} + \mathbf{v})\ \forall \mathbf{x}$.*
- *For any θ_1, θ_2, there exists θ_0 such that $F^{\theta_0}(t) \geq \max\left\{F^{\theta_1}(t), F^{\theta_2}(t)\right\}\ \forall t \geq 0$ and $p_{\theta_0}(\mathbf{0}) = \max\left\{p_{\theta_1}(\mathbf{0}), p_{\theta_2}(\mathbf{0})\right\}$.*

- $\exists \tau > 0$ *such that* $\forall i \in [n]$ $\forall \theta \notin B_{\theta^*}(\tau)$, $p_\theta(\mathbf{x}_i) < p_{\theta*}(\mathbf{x}_i)$, *where* $B_{\theta^*}(\tau)$ *denotes the ball centred at* θ^* *of radius* τ.
- $\Psi(z)$ *is differentiable everywhere.*
- *For all* θ, *if* $\theta \neq \theta^*$, *there exists* $j \in [d]$ *such that*
$$\left\langle \begin{pmatrix} \frac{\Psi'(p_\theta(\mathbf{x}_1))p_\theta(\mathbf{x}_1)}{\Psi'(p_{\theta^*}(\mathbf{x}_1))p_{\theta^*}(\mathbf{x}_1)} \\ \vdots \\ \frac{\Psi'(p_\theta(\mathbf{x}_n))p_\theta(\mathbf{x}_n)}{\Psi'(p_{\theta^*}(\mathbf{x}_n))p_{\theta^*}(\mathbf{x}_n)} \end{pmatrix}, \begin{pmatrix} \nabla_\theta \left(\log p_\theta(\mathbf{x}_1)\right)_j \\ \vdots \\ \nabla_\theta \left(\log p_\theta(\mathbf{x}_n)\right)_j \end{pmatrix} \right\rangle \neq 0.$$

Then,

$$\arg\min_\theta \sum_{i=1}^n \frac{\mathbb{E}\left[R_i^\theta\right]}{\Psi'(p_{\theta^*}(\mathbf{x}_i))p_{\theta^*}(\mathbf{x}_i)} = \arg\max_\theta \sum_{i=1}^n \log p_\theta(\mathbf{x}_i)$$

Furthermore, if $p_{\theta^*}(\mathbf{x}_1) = \cdots = p_{\theta^*}(\mathbf{x}_n)$, *then,*

$$\arg\min_\theta \sum_{i=1}^n \mathbb{E}\left[R_i^\theta\right] = \arg\max_\theta \sum_{i=1}^n \log p_\theta(\mathbf{x}_i)$$

Proof. Pick an arbitrary $i \in [n]$. We first prove a few basic facts.

By the second property of P_θ, $\forall \theta$ $\exists \theta'$ such that $p_\theta(\mathbf{u}) = p_{\theta'}(\mathbf{u} - \mathbf{x}_i)$ $\forall \mathbf{u}$. In particular, $p_\theta(\mathbf{x}_i) = p_{\theta'}(\mathbf{x}_i - \mathbf{x}_i) = p_{\theta'}(\mathbf{0})$. Let F_i^θ be as defined in Lemma 15.

$$\begin{aligned} F_i^\theta(t) &= \Pr\left(\tilde{r}_i^\theta \leq t\right) = \Pr\left(\left\|\tilde{\mathbf{x}}_1^\theta - \mathbf{x}_i\right\|_2 \leq \sqrt{t}\right) \\ &= \int_{B_{\mathbf{x}_i}(\sqrt{t})} p_\theta(\mathbf{u}) d\mathbf{u} = \int_{B_{\mathbf{x}_i}(\sqrt{t})} p_{\theta'}(\mathbf{u} - \mathbf{x}_i) d\mathbf{u} \\ &= \int_{B_{\mathbf{0}}(\sqrt{t})} p_{\theta'}(\mathbf{u}) d\mathbf{u} = \Pr\left(\tilde{r}^{\theta'} \leq t\right) = F^{\theta'}(t) \end{aligned}$$

Similarly, $\forall \theta'$ $\exists \theta$ such that $p_{\theta'}(\mathbf{u}) = p_\theta(\mathbf{u} + \mathbf{x}_i)$ $\forall \mathbf{u}$. In particular, $p_{\theta'}(\mathbf{0}) = p_\theta(\mathbf{0} + \mathbf{x}_i) = p_\theta(\mathbf{x}_i)$.

$$\begin{aligned} F^{\theta'}(t) &= \Pr\left(\tilde{r}^{\theta'} \leq t\right) = \int_{B_{\mathbf{0}}(\sqrt{t})} p_{\theta'}(\mathbf{u}) d\mathbf{u} \\ &= \int_{B_{\mathbf{0}}(\sqrt{t})} p_\theta(\mathbf{u} + \mathbf{x}_i) d\mathbf{u} = \int_{B_{\mathbf{x}_i}(\sqrt{t})} p_\theta(\mathbf{u}) d\mathbf{u} \\ &= \Pr\left(\left\|\tilde{\mathbf{x}}_1^\theta - \mathbf{x}_i\right\|_2 \leq \sqrt{t}\right) = \Pr\left(\tilde{r}_i^\theta \leq t\right) = F_i^\theta(t) \end{aligned}$$

Let θ_1, θ_2 be arbitrary. The facts above imply that there exist θ_1' and θ_2' such that $F_i^{\theta_1}(t) = F^{\theta_1'}(t)$, $F_i^{\theta_2}(t) = F^{\theta_2'}(t)$, $p_{\theta_1}(\mathbf{x}_i) = p_{\theta_1'}(\mathbf{0})$ and $p_{\theta_2}(\mathbf{x}_i) = p_{\theta_2'}(\mathbf{0})$.

By the third property of P_θ, let θ_0' be such that $F^{\theta_0'}(t) \geq \max\left\{F^{\theta_1'}(t), F^{\theta_2'}(t)\right\}$ $\forall t \geq 0$ and $p_{\theta_0'}(\mathbf{0}) = \max\left\{p_{\theta_1'}(\mathbf{0}), p_{\theta_2'}(\mathbf{0})\right\}$. By the facts above, it follows that there exists θ_0 such that $F^{\theta_0'}(t) = F_i^{\theta_0}(t)$ and $p_{\theta_0'}(\mathbf{0}) = p_{\theta_0}(\mathbf{x}_i)$.

So, we can conclude that for any θ_1, θ_2, there exists θ_0 such that $F_i^{\theta_0}(t) \geq \max\left\{F_i^{\theta_1}(t), F_i^{\theta_2}(t)\right\}\ \forall t \geq 0$ and $p_{\theta_0}(\mathbf{x}_i) = \max\left\{p_{\theta_1}(\mathbf{x}_i), p_{\theta_2}(\mathbf{x}_i)\right\}$.

By Lemma 15, $\Psi_i(z) = \min_\theta \left\{\mathbb{E}\left[R_i^\theta\right] | p_\theta(\mathbf{x}_i) = z\right\}$ is strictly decreasing.

Consider any θ. By the facts above, there exists θ' such that $p_\theta(\mathbf{x}_i) = p_{\theta'}(\mathbf{0})$ and $F_i^\theta(t) = F^{\theta'}(t)\ \forall t$. Therefore,

$$\begin{aligned}
\mathbb{E}\left[R_i^\theta\right] &= \int_0^\infty \Pr\left(R_i^\theta > t\right) dt \\
&= \int_0^\infty \left(\Pr\left(\left\|\tilde{\mathbf{x}}_1^\theta - \mathbf{x}_i\right\|_2^2 > t\right)\right)^m dt \\
&= \int_0^\infty \left(1 - F_i^\theta(t)\right)^m dt \\
&= \int_0^\infty \left(1 - F^{\theta'}(t)\right)^m dt \\
&= \int_0^\infty \Pr\left(R^{\theta'} > t\right) dt \\
&= \mathbb{E}\left[R^{\theta'}\right]
\end{aligned}$$

So, $\forall z$

$$\begin{aligned}
\Psi_i(z) &= \min_\theta \left\{\mathbb{E}\left[R_i^\theta\right] | p_\theta(\mathbf{x}_i) = z\right\} \\
&= \min_{\theta'} \left\{\mathbb{E}\left[R^{\theta'}\right] | p_{\theta'}(\mathbf{0}) = z\right\} \\
&= \Psi(z)
\end{aligned}$$

Because $\Psi_i(\cdot)$ is strictly decreasing, $\Psi(\cdot)$ is also strictly decreasing.

We would like to apply Lemma 13, with $f_i(\theta) = -\log p_\theta(\mathbf{x}_i)\ \forall i \in [n]$ and $\Phi(y) = \Psi(\exp(-y))$. By the first property of P_θ, $p_\theta(\cdot)$ is differentiable w.r.t. θ and so $f_i(\theta)$ is differentiable for all i. By the fifth property of P_θ, $\Psi(\cdot)$ is differentiable and so $\Phi(\cdot)$ is differentiable. Since $y \mapsto \exp(-y)$ is strictly decreasing and $\Psi(\cdot)$ is strictly decreasing, $\Phi(\cdot)$ is strictly increasing. Since there is a unique maximum likelihood solution θ^*, $\min_\theta \sum_{i=1}^n f_i(\theta) = \max_\theta \sum_{i=1}^n \log p_\theta(\mathbf{x}_i)$ exists and has a unique minimizer. By the fourth property of P_θ, the first condition of Lemma 13 is satisfied. By the sixth property of P_θ, the second condition of Lemma 13 is satisfied. Since all conditions are satisfied, we apply Lemma 13 and conclude

that

$$\begin{aligned}\min_\theta \sum_{i=1}^n w_i \Phi(f_i(\theta)) &= \min_\theta \sum_{i=1}^n w_i \Psi(p_\theta(\mathbf{x}_i)) \\ &= \min_\theta \sum_{i=1}^n w_i \Psi_i(p_\theta(\mathbf{x}_i)) \\ &= \min_\theta \sum_{i=1}^n \frac{\mathbb{E}\left[R_i^\theta\right]}{\Psi'(p_{\theta^*}(\mathbf{x}_i))p_{\theta^*}(\mathbf{x}_i)}\end{aligned}$$

exists and has a unique minimizer. Furthermore,

$$\begin{aligned}\arg\min_\theta \sum_{i=1}^n \frac{\mathbb{E}\left[R_i^\theta\right]}{\Psi'(p_{\theta^*}(\mathbf{x}_i))p_{\theta^*}(\mathbf{x}_i)} &= \arg\min_\theta \sum_{i=1}^n -\log p_\theta(\mathbf{x}_i) \\ &= \arg\max_\theta \sum_{i=1}^n \log p_\theta(\mathbf{x}_i)\end{aligned}$$

If $p_\theta(\mathbf{x}_1) = \cdots p_\theta(\mathbf{x}_n)$, then $w_1 = \cdots = w_n$, and so
$\arg\min_\theta \sum_{i=1}^n w_i \mathbb{E}\left[R_i^\theta\right] = \arg\min_\theta \sum_{i=1}^n \mathbb{E}\left[R_i^\theta\right] = \arg\max_\theta \sum_{i=1}^n \log p_\theta(\mathbf{x}_i)$. □

Bibliography

[1] Evangelos Anagnostopoulos, Ioannis Z Emiris, and Ioannis Psarros. "Low-quality dimension reduction and high-dimensional approximate nearest neighbor". In: *31st International Symposium on Computational Geometry (SoCG 2015)*. 2015, pp. 436–450.

[2] Alexandr Andoni and Piotr Indyk. "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions". In: *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*. IEEE. 2006, pp. 459–468.

[3] Alexandr Andoni and Ilya Razenshteyn. "Optimal data-dependent hashing for approximate near neighbors". In: *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*. ACM. 2015, pp. 793–801.

[4] Marcin Andrychowicz et al. "Learning to learn by gradient descent by gradient descent". In: *arXiv preprint arXiv:1606.04474* (2016).

[5] Martin Arjovsky and Léon Bottou. "Towards principled methods for training generative adversarial networks". In: *arXiv preprint arXiv:1701.04862* (2017).

[6] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein generative adversarial networks". In: *International Conference on Machine Learning*. 2017, pp. 214–223.

[7] Sanjeev Arora and Yi Zhang. "Do GANs actually learn the distribution? an empirical study". In: *arXiv preprint arXiv:1706.08224* (2017).

[8] Sanjeev Arora et al. "Generalization and equilibrium in generative adversarial nets (GANs)". In: *arXiv preprint arXiv:1703.00573* (2017).

[9] Kenneth Joseph Arrow et al. "Studies in linear and non-linear programming". In: (1958).

[10] Sunil Arya and David M Mount. "Approximate Nearest Neighbor Queries in Fixed Dimensions." In: *SODA*. Vol. 93. 1993, pp. 271–280.

[11] Sunil Arya et al. "An optimal algorithm for approximate nearest neighbor searching fixed dimensions". In: *Journal of the ACM (JACM)* 45.6 (1998), pp. 891–923.

[12] Leonard E Baum and Ted Petrie. "Statistical inference for probabilistic functions of finite state Markov chains". In: *The annals of mathematical statistics* 37.6 (1966), pp. 1554–1563.

[13] Jonathan Baxter et al. *NIPS 1995 Workshop on Learning to Learn: Knowledge Consolidation and Transfer in Inductive Systems*. https://web.archive.org/web/20000618135816/http://www.cs.cmu.edu/afs/cs.cmu.edu/user/caruana/pub/transfer.html. Accessed: 2015-12-05. 1995.

[14] Rudolf Bayer. "Symmetric binary B-trees: Data structure and maintenance algorithms". In: *Acta informatica* 1.4 (1972), pp. 290–306.

[15] Ehsan Behnam, Michael S Waterman, and Andrew D Smith. "A geometric interpretation for local alignment-free sequence comparison". In: *Journal of Computational Biology* 20.7 (2013), pp. 471–485.

[16] Y Bengio, S Bengio, and J Cloutier. "Learning a synaptic learning rule". In: *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*. Vol. 2. IEEE. 1991, 969–vol.

[17] Yoshua Bengio. "Gradient-based optimization of hyperparameters". In: *Neural computation* 12.8 (2000), pp. 1889–1900.

[18] Yoshua Bengio and Samy Bengio. "Modeling high-dimensional discrete data with multi-layer neural networks". In: *Advances in Neural Information Processing Systems*. 2000, pp. 400–406.

[19] Yoshua Bengio et al. "Better Mixing via Deep Representations." In: *ICML (1)*. 2013, pp. 552–560.

[20] Yoshua Bengio et al. "Deep generative stochastic networks trainable by backprop". In: *International Conference on Machine Learning*. 2014, pp. 226–234.

[21] Jon Louis Bentley. "Multidimensional binary search trees used for associative searching". In: *Communications of the ACM* 18.9 (1975), pp. 509–517.

[22] Stefan Berchtold, Daniel A. Keim, and Hans-peter Kriegel. "The X-tree: An Index Structure for High-Dimensional Data". In: *Very Large Data Bases*. 1996, pp. 28–39.

[23] Stefan Berchtold et al. "Fast nearest neighbor search in high-dimensional space". In: *Data Engineering, 1998. Proceedings., 14th International Conference on*. IEEE. 1998, pp. 209–218.

[24] James S Bergstra et al. "Algorithms for hyper-parameter optimization". In: *Advances in Neural Information Processing Systems*. 2011, pp. 2546–2554.

[25] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 281–305.

[26] Julian Besag. "Statistical analysis of non-lattice data". In: *The statistician* (1975), pp. 179–195.

[27] Alina Beygelzimer, Sham Kakade, and John Langford. "Cover trees for nearest neighbor". In: *Proceedings of the 23rd International Conference on Machine Learning*. ACM. 2006, pp. 97–104.

[28] Gérard Biau et al. "A weighted k-nearest neighbor density estimate for geometric inference". In: *Electronic Journal of Statistics* 5 (2011), pp. 204–237.

[29] Christopher M Bishop, Markus Svensén, and Christopher KI Williams. "GTM: The generative topographic mapping". In: *Neural computation* 10.1 (1998), pp. 215–234.

[30] Matthieu Bray et al. "3D Hand Tracking by Rapid Stochastic Gradient Descent using a Skinning Model". In: *1st European Conference on Visual Media Production (CVMP)*. Citeseer. 2004.

[31] Pavel B Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. "Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results". In: *Machine Learning* 50.3 (2003), pp. 251–277.

[32] Pavel Brazdil et al. *Metalearning: applications to data mining*. Springer Science & Business Media, 2008.

[33] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. "Importance weighted autoencoders". In: *arXiv preprint arXiv:1509.00519* (2015).

[34] Qifeng Chen and Vladlen Koltun. "Photographic image synthesis with cascaded refinement networks". In: *IEEE International Conference on Computer Vision (ICCV)*. Vol. 1. 2. 2017, p. 3.

[35] Kenneth L Clarkson. "Nearest neighbor queries in metric spaces". In: *Discrete & Computational Geometry* 22.1 (1999), pp. 63–93.

[36] Kenneth L Clarkson. "Nearest-neighbor searching and metric space dimensions". In: *Nearest-neighbor methods for learning and vision: theory and practice* (2006), pp. 15–59.

[37] Marius Cordts et al. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[38] Nichael Lynn Cramer. "A representation for the adaptive generation of simple sequential programs". In: *Proceedings of the First International Conference on Genetic Algorithms*. 1985, pp. 183–187.

[39] Christian Daniel, Jonathan Taylor, and Sebastian Nowozin. "Learning Step Size Controllers for Robust Neural Network Training". In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[40] Sanjoy Dasgupta and Yoav Freund. "Random projection trees and low dimensional manifolds". In: *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*. ACM. 2008, pp. 537–546.

[41] Sanjoy Dasgupta and Kaushik Sinha. "Randomized partition trees for nearest neighbor search". In: *Algorithmica* 72.1 (2015), pp. 237–263.

[42] Mayur Datar et al. "Locality-sensitive hashing scheme based on p-stable distributions". In: *Proceedings of the twentieth annual symposium on Computational geometry*. ACM. 2004, pp. 253–262.

[43] Peter J Diggle and Richard J Gratton. "Monte Carlo methods of inference for implicit statistical models". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1984), pp. 193–227.

[44] Justin Domke. "Generic Methods for Optimization-Based Modeling." In: *AISTATS*. Vol. 22. 2012, pp. 318–326.

[45] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. "Adversarial feature learning". In: *arXiv preprint arXiv:1605.09782* (2016).

[46] Alexey Dosovitskiy and Thomas Brox. "Generating images with perceptual similarity metrics based on deep networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 658–666.

[47] Vincent Dumoulin et al. "Adversarially learned inference". In: *arXiv preprint arXiv:1606.00704* (2016).

[48] Gintare Karolina Dziugaite, Daniel M Roy, and Zoubin Ghahramani. "Training generative neural networks via maximum mean discrepancy optimization". In: *arXiv preprint arXiv:1505.03906* (2015).

[49] Francis Ysidro Edgeworth. "On the probable errors of frequency-constants". In: *Journal of the Royal Statistical Society* 71.2 (1908), pp. 381–397.

[50] Ahmed Eldawy and Mohamed F Mokbel. "SpatialHadoop: A MapReduce framework for spatial data". In: *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. IEEE. 2015, pp. 1352–1363.

[51] Brian S Everitt. *Mixture Distributions - I*. Wiley Online Library, 1985.

[52] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. "Initializing Bayesian Hyperparameter Optimization via Meta-Learning." In: *AAAI*. 2015, pp. 1128–1135.

[53] Chelsea Finn et al. "Learning Visual Feature Spaces for Robotic Manipulation with Deep Spatial Autoencoders". In: *arXiv preprint arXiv:1509.06113* (2015).

[54] Ronald A Fisher. "On an absolute criterion for fitting frequency curves". In: *Messenger of Mathematics* 41 (1912), pp. 155–160.

[55] Evelyn Fix and Joseph L Hodges Jr. *Discriminatory analysis-nonparametric discrimination: consistency properties*. Tech. rep. California Univ Berkeley, 1951.

[56] Jie Fu et al. "Deep Q-Networks for Accelerating the Training of Deep Neural Networks". In: *arXiv preprint arXiv:1606.01467* (2016).

[57] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[58] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines". In: *arXiv preprint arXiv:1410.5401* (2014).

[59] Karol Gregor and Yann LeCun. "Learning fast approximations of sparse coding". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 399–406.

[60] Arthur Gretton et al. "A kernel method for the two-sample-problem". In: *Advances in neural information processing systems*. 2007, pp. 513–520.

[61] Leo J Guibas and Robert Sedgewick. "A dichromatic framework for balanced trees". In: *Foundations of Computer Science, 1978., 19th Annual Symposium on*. IEEE. 1978, pp. 8–21.

[62] Michael U Gutmann et al. "Likelihood-free inference via classification". In: *arXiv preprint arXiv:1407.4981* (2014).

[63] Michael Gutmann and Aapo Hyvärinen. "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models." In: *AISTATS*. Vol. 1. 2. 2010, p. 6.

[64] Antonin Guttman. "R-trees: a dynamic index structure for spatial searching". In: *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*. 1984, pp. 47–57.

[65] Weiqiao Han, Sergey Levine, and Pieter Abbeel. "Learning Compound Multi-Step Controllers under Unknown Dynamics". In: *International Conference on Intelligent Robots and Systems*. 2015.

[66] Samantha Hansen. "Using Deep Q-Learning to Control Optimization Hyperparameters". In: *arXiv preprint arXiv:1602.04062* (2016).

[67] Martin Heusel et al. "Gans trained by a two time-scale update rule converge to a local nash equilibrium". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6626–6637.

[68] Geoffrey E Hinton. "Training products of experts by minimizing contrastive divergence". In: *Neural computation* 14.8 (2002), pp. 1771–1800.

[69] Geoffrey E Hinton and Terrence J Sejnowski. "Learning and releaming in boltzmann machines". In: *Parallel distributed processing: Explorations in the microstructure of cognition* 1.282-317 (1986), p. 2.

[70] R Devon Hjelm et al. "Boundary-seeking generative adversarial networks". In: *arXiv preprint arXiv:1702.08431* (2017).

[71] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[72] Sepp Hochreiter, A Steven Younger, and Peter R Conwell. "Learning to learn using gradient descent". In: *International Conference on Artificial Neural Networks*. Springer. 2001, pp. 87–94.

[73] Michael E Houle and Michael Nett. "Rank-based similarity search: Reducing the dimensional dependence". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 37.1 (2015), pp. 136–150.

[74] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. "Sequential model-based optimization for general algorithm configuration". In: *Learning and Intelligent Optimization*. Springer, 2011, pp. 507–523.

[75] Aapo Hyvärinen. "Estimation of non-normalized statistical models by score matching". In: *Journal of Machine Learning Research* 6.Apr (2005), pp. 695–709.

[76] Piotr Indyk and Rajeev Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality". In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. ACM. 1998, pp. 604–613.

[77] Phillip Isola et al. "Image-To-Image Translation With Conditional Adversarial Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1125–1134.

[78] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. "Product quantization for nearest neighbor search". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 33.1 (2011), pp. 117–128.

[79] Michael I Jordan et al. "An introduction to variational methods for graphical models". In: *Machine learning* 37.2 (1999), pp. 183–233.

[80] David R Karger and Matthias Ruhl. "Finding nearest neighbors in growth-restricted metrics". In: *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*. ACM. 2002, pp. 741–750.

[81] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[82] Robert Krauthgamer and James R Lee. "Navigating nets: simple algorithms for proximity search". In: *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics. 2004, pp. 798–807.

[83] Alex Krizhevsky and Geoffrey Hinton. "Learning multiple layers of features from tiny images". In: *Technical report, University of Toronto* (2009).

[84] Hugo Larochelle and Iain Murray. "The neural autoregressive distribution estimator". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, pp. 29–37.

[85] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[86] Sergey Levine and Pieter Abbeel. "Learning neural network policies with guided policy search under unknown dynamics". In: *Advances in Neural Information Processing Systems*. 2014, pp. 1071–1079.

[87] Sergey Levine, Nolan Wagener, and Pieter Abbeel. "Learning Contact-Rich Manipulation Skills with Guided Policy Search". In: *arXiv preprint arXiv:1501.05611* (2015).

[88] Sergey Levine et al. "End-to-End Training of Deep Visuomotor Policies". In: *arXiv preprint arXiv:1504.00702* (2015).

[89] Ke Li and Jitendra Malik. "Fast k-nearest neighbour search via Dynamic Continuous Indexing". In: *International Conference on Machine Learning*. 2016, pp. 671–679.

[90] Ke Li and Jitendra Malik. "Fast k-nearest neighbour search via Dynamic Continuous Indexing". In: *International Conference on Machine Learning*. 2016, pp. 671–679.

[91] Ke Li and Jitendra Malik. "Fast k-Nearest Neighbour Search via Prioritized DCI". In: *International Conference on Machine Learning*. 2017, pp. 2081–2090.

[92] Yujia Li, Kevin Swersky, and Rich Zemel. "Generative moment matching networks". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 2015, pp. 1718–1727.

[93] Yujia Li et al. "Dualing GANs". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5611–5621.

[94] Percy Liang, Michael I Jordan, and Dan Klein. "Learning programs: A hierarchical Bayesian approach". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 639–646.

[95] Ting Liu et al. "An investigation of practical approximate nearest neighbor algorithms". In: *Advances in Neural Information Processing Systems*. 2004, pp. 825–832.

[96] David JC MacKay. "Bayesian neural networks and density networks". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 354.1 (1995), pp. 73–80.

[97] Dougal Maclaurin, David Duvenaud, and Ryan P Adams. "Gradient-based hyperparameter optimization through reversible learning". In: *arXiv preprint arXiv:1502.03492* (2015).

[98] Stefan Meiser. "Point location in arrangements of hyperplanes". In: *Information and Computation* 106.2 (1993), pp. 286–303.

[99] Marvin Minsky and Seymour Papert. "Perceptrons: an introduction to computational geometry". In: (1969), p. 222.

[100] Shakir Mohamed and Balaji Lakshminarayanan. "Learning in implicit generative models". In: *arXiv preprint arXiv:1610.03483* (2016).

[101] Alfred Müller. "Integral probability metrics and their generating classes of functions". In: *Advances in Applied Probability* 29.2 (1997), pp. 429–443.

[102] Radford M Neal. "Connectionist learning of belief networks". In: *Artificial intelligence* 56.1 (1992), pp. 71–113.

[103] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. "f-GAN: Training generative neural samplers using variational divergence minimization". In: *Advances in Neural Information Processing Systems*. 2016, pp. 271–279.

[104] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel recurrent neural networks". In: *arXiv preprint arXiv:1601.06759* (2016).

[105] Michael T Orchard. "A fast nearest-neighbor search algorithm". In: *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*. IEEE. 1991, pp. 2297–2300.

[106] Loïc Paulevé, Hervé Jégou, and Laurent Amsaleg. "Locality sensitive hashing: A comparison of hash function types and querying mechanisms". In: *Pattern Recognition Letters* 31.11 (2010), pp. 1348–1358.

[107] William Pugh. "Skip lists: a probabilistic alternative to balanced trees". In: *Communications of the ACM* 33.6 (1990), pp. 668–676.

[108] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[109] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic backpropagation and variational inference in deep latent Gaussian models". In: *International Conference on Machine Learning*. 2014.

[110] Stephan R. Richter et al. "Playing for Data: Ground Truth from Computer Games". In: *European Conference on Computer Vision (ECCV)*. Ed. by Bastian Leibe et al. Vol. 9906. LNCS. Springer International Publishing, 2016, pp. 102–118.

[111] Paul L Ruvolo, Ian Fasel, and Javier R Movellan. "Optimization on a budget: A reinforcement learning approach". In: *Advances in Neural Information Processing Systems*. 2009, pp. 1385–1392.

[112] Tim Salimans et al. "Improved techniques for training GANs". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2234–2242.

[113] Jürgen Schmidhuber. "Learning factorial codes by predictability minimization". In: *Neural Computation* 4.6 (1992), pp. 863–879.

[114] Jürgen Schmidhuber. "Optimal ordered problem solver". In: *Machine Learning* 54.3 (2004), pp. 211–254.

[115] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[116] Mathieu Sinn and Ambrish Rawat. "Non-parametric estimation of Jensen-Shannon Divergence in Generative Adversarial Network training". In: *arXiv preprint arXiv:1705.09199* (2017).

[117] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems*. 2012, pp. 2951–2959.

[118] Pablo Sprechmann et al. "Supervised sparse analysis and synthesis operators". In: *Advances in Neural Information Processing Systems*. 2013, pp. 908–916.

[119] Kevin Swersky, Jasper Snoek, and Ryan P Adams. "Multi-task bayesian optimization". In: *Advances in neural information processing systems*. 2013, pp. 2004–2012.

[120] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[121] Zhuowen Tu. "Learning generative models via discriminative approaches". In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE. 2007, pp. 1–8.

[122] Ricardo Vilalta and Youssef Drissi. "A perspective view and survey of meta-learning". In: *Artificial Intelligence Review* 18.2 (2002), pp. 77–95.

[123] Ting-Chun Wang et al. "High-resolution image synthesis and semantic manipulation with conditional gans". In: *arXiv preprint arXiv:1711.11585* (2017).

[124] Xintao Wang et al. "ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks". In: *CoRR* abs/1809.00219 (2018).

[125] Yair Weiss, Antonio Torralba, and Rob Fergus. "Spectral hashing". In: *Advances in Neural Information Processing Systems*. 2009, pp. 1753–1760.

[126] Max Welling and Geoffrey Hinton. "A new learning algorithm for mean field Boltzmann machines". In: *Artificial Neural Networks - ICANN 2002* (2002), pp. 82–82.

[127] Yuhuai Wu et al. "On the quantitative analysis of decoder-based generative models". In: *arXiv preprint arXiv:1611.04273* (2016).

[128] Richard Zhang et al. "The unreasonable effectiveness of deep features as a perceptual metric". In: *arXiv preprint* (2018).

[129] Junbo Zhao, Michael Mathieu, and Yann LeCun. "Energy-based generative adversarial network". In: *arXiv preprint arXiv:1609.03126* (2016).

[130] Jun-Yan Zhu et al. "Toward Multimodal Image-to-Image Translation". In: *CoRR* abs/1711.11586 (2017). arXiv: 1711.11586. URL: http://arxiv.org/abs/1711.11586.

[131] Jun-Yan Zhu et al. "Toward multimodal image-to-image translation". In: *Advances in Neural Information Processing Systems*. 2017, pp. 465–476.

[132] Jun-Yan Zhu et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks". In: *arXiv preprint arXiv:1703.10593* (2017).